sinclair COMPUTER

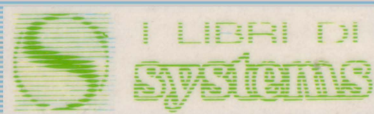
PRESENTA

Tutti i segreti dello SPECTRUM

4 passi nella ROM: come utilizzare le routine più importanti del sistema operativo

a cura di **Giovanni Mellina**

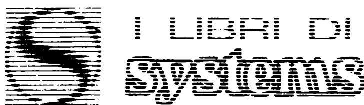




Tutti i segreti dello SPECTRUM

4 passi nella ROM: come utilizzare le routine più importanti del sistema operativo

a cura di G. Mellina



sommario

appendici

Le routines più utili

Premessa
Le aree di sistema in RAM

Si possono distinguere in due categorie:
aree fisse: *display-file*, attributi, *printer buffer*, variabili di sistema; sono le aree il cui indirizzo e la cui ampiezza non subiscono mai variazioni;

aree dinamiche: *microdrive maps, channel informations,* area basic, variabili, *edit-area, work space* o *input-area, calculator stack, spare space, machine* e GOSUB stack; sono le aree dinamiche, destinate a espandersi e restringersi secondo necessità.

Il loro indirizzo, modificandosi dinamicamente in conseguenza di variazioni di ampiezza di aree più basse, è mantenuto aggiornato e reperibile nelle variabili di sistema (da 23627 a 23654).

La loro ampiezza, essendo anch'essa dinamica, è ricavabile per differenza tra gli indirizzi dell'area interessata e di quella immediatamente più alta.

In particolare lo *spare space*, il cui inizio è indicato dalla variabile STKEND, è il serbatoio di riserva, cioè la memoria disponibile, da cui attingere per poter espandere le altre aree.

Non esiste invece una variabile indicante l'indirizzo di *machine stack*: è reperibile solo in codice macchina (registro SP).

La tastiera

KEY-SCAN (CALL dec 703 hex 02BF)

esegue la lettura di tutta la tastiera e imposta le variabili KSTATE e LASTK in funzione dei tasti premuti.

reg. out:
A = CODE tasti premuti = var. KSTATE (23556)
BC - DE - HL modificati
IX non modificato

KEY - INPUT (CALL dec 4264 hex 10A8)

ottiene il CODE dell'ultimo tasto premuto, acquisendolo dalla variabile LASTK; inoltre, se il bit 5 di TVFLAG è ON (SET 5, (IY + 2)), esegue CLS della *screen edit area* (righe 23-24).

reg. out:

A = CODE tasti premuti = variabile LASTK se vi è stata immissione, altrimenti non modificato

BC - DE - HL modificati solo con bit 5 di TVFLAG ON

IX non modificato

flags out:

C se vi è stata immissione; NZ se il carattere è producibile (CODE maggiore di 31)

NC e Z se non vi è stata immissione

KEY-WAIT (CALL dec 5598 hex 15DE)

ritorna solo dopo pressione di un qualsiasi tasto; equivalente a un PAUSE 0. Se il bit 5 di TVFLAG è ON, esegue CLS della *screen edit area* (righe 23-24)

Prima di invocarla è necessario aprire il flusso relativo al canale K (vedi il paragrafo *La stampa*) eseguendo LD A, 1 : CALL 5633.

reg. out:

A = CODE tasto premuto = variabile LASTK;

altri non modificati

BREAK (CALL dec 8020 hex 1F54)

è la routine che testa la pressione dei tasti CAPS S. e SPACE.

reg. out:

A modificato

altri non modificati

flags out:

NC se è stato richiesto il break

variabili:

KSTATE (byte 23556) indica sempre il CODE maiuscolo corrispondente al tasto premuto (anche con CAPS S. non usato). oppure 255 se non vi è stata immissione.

LASTK (23560) indica sempre il CODE corrispondente all'ultima combinazione di tasti premuti.

FLAGS (23611) indica se vi è stata pressione di un tasto (bit 5 ON), o nessuna immissione (bit 5 OFF).

FLAGS 2 (23658): impostando ON il bit 3 si forza il CAPS S. mode per le successive immissioni.

MODE (23617) indica il modo di immissione in corso (lettera del cursore):

1 = "E"
2 = "G"
0 + FLAGS bit 2 OFF = "K"
0 + FLAGS bit 2 ON + FLAGS 2 bit 3 OFF = "L"
0 + FLAGS bit 2 ON + FLAGS 2 bit 3 ON = "C"

La cassetta (tape)

NO - HDR - LOAD (CALL dec 1366 hex 0556)
NO - HDR - SAVE (CALL dec 1218 hex 04C2)
eseguono LOAD e SAVE di files senza *header*.

reg. in:
IX = indirizzo dell'area di memoria da cui iniziare LOAD o SAVE
DE = lunghezza del file (bytes)
A = 255 indicante *data-block*

flags in: C = LOAD, NC = SAVE
reg. out: tutti modificati

flags out: (se LOAD):
C = OK
NC = tape loading error
NC e Z = *time up* o *No parity-check*
NC e NZ = break o *No match file-type*

Nota: il LOAD possiede anche un entry - point a dec 2048 (hex 0800), da cui si rientra solo se OK (flag C), altrimenti si ha errore R.

LOAD (CALL dec 1366 e 2048 hex 0556 e 0800) eseguono il caricamento di una normale registrazione, completa di header.

Si può utilizzare la seguente routine:

```
9C40                5        ORG  40000
9C40 DD2128A0      10 tld    LD   IX,41000
     ;17 bytes destinati all'header
9C44 DDE5          20        PUSH IX
9C46 111100        30        LD   DE,17
9C49 AF            40        XOR  A
     ;A=0 indicante header
9C4A 37            50        SCF  ;flag C indic
     ante LOAD
9C4B CD5605        60        CALL 1366
     ;load header
9C4E DDE1          70        POP  IX
9C50 30EE          80        JR   NC,tld
9C52 DD5E0B        90        LD   E,(IX+11)
     ;lunghezza data-block
9C55 DD560C       100        LD   D,(IX+12)
9C58 DD7E00       110        LD   A,(IX+0)
     ;file type
9C5B DD2A535C     120        LD   IX,(23635)
     ;indir.area basic
9C5F EE03         130        XOR  3
9C61 2009         140        JR   NZ,noc
     ;salta oltre se file type<>3
9C63 DD6E0D       150        LD   L,(IX+13)
     ;indir.inizio CODE
9C66 DD660E       160        LD   H,(IX+14)
```

```
9C69 E5          170         PUSH HL
9C6A DDE1        180         POP  IX
9C6C 37          190 noc     SCF  ;flag C indic
     ante LOAD
9C6D CD0008      200         CALL 2048
     ;load data-block
9C70             210         END
```

SAVE (CALL dec 2416 hex 0970)
esegue una normale registrazione completa di header.

reg. in:
IX = indirizzo dell'area (17 bytes) contenente header
HL = indirizzo dell'area da cui prelevare i dati
Nota: l'header deve essere predisposto completo di tutte le informazioni (tipo di file, nome, lunghezza *data-block*) necessarie al SAVE.

Header: contiene
byte
0 = 0 (progr. basic), 1 (num. array), 2 (char. array), 3 (bytes)
1 ÷ 10 = nome (255 in pos. 1 se nullo)
11 ÷ 12 = lunghezza CODE o array o progr. basic + variabili
13 ÷ 14 = autorun LINE (progr.) o start address (CODE)
15 ÷ 16 = lunghezza del programma (solo area basic)
14 = nome se array
file: ogni file (*header* o *data-block*) viene registrato con 2 bytes aggiunti, non conteggiati nella lunghezza: uno in testa, indicante il *file-type* 0 (header) o 255 (data-block) ed uno in coda, indicante il valore di *parity-check* per testarne la corretta lettura con LOAD.

Il beeper

BEEP-1 (CALL 949 hex 03B5)
emette la nota acustica corrispondente ai valori contenuti

nei registri HL e DE.

reg. in:
HL = valore funzione della frequenza
DE = valore funzione della durata

reg. out:
A e DE = 0
altri modificati

Note: Il valore in HL si ottiene da:
437500 / (frequenza) — 30.125
Il valore in DE è direttamente proporzionale alla frequenza e inversamente proporzionale al valore di HL; si ottiene da:
*frequenza * durata*
Per cui un
BEEP 1,0 (frequenza DO = 261.63 Hz)
si ottiene con:
HL = 437500 / 261.63 - 30.125 = 1642
DE = 261.63 * 1 = 262
Conoscendo la frequenza di una nota, si ricava quella della seminota superiore con:
frequenza * 1.05946315
e della seminota inferiore con:
frequenza / 1.05946315
Il LA della scala di DO (diapason) ha una frequenza di 220 Hz.

BEEP-2 (CALL 1016 hex 03F8)
è la routine che esegue l'analogo comando basic prelevando i valori di tono e durata dal *calculator stack;* per esempio, un BEEP 1,0 si ottiene con:

```
9C40                5         ORG  40000
9C40 3E01           10        LD   A,1
     ;durata
9C42 CD282D         20        CALL 11560
     ;carica A nel calc.stack
```

```
9C45 3E00          30          LD   A,0
     ;tono
9C47 CD282D        40          CALL 11560
     ;carica A nel calc.stack
9C4A CDF803        50          CALL 1016
     ;beep
9C4D C9            60          RET
```

reg. out:
A e DE = 0
altri modificati

Nota: il *calculator stack* e le routines che lo gestiscono sono specificatamente trattati al capitolo *Il calcolatore F.P.*

La stampa

OPEN (CALL dec. 5633 hex 1601)
apre un flusso (stream) verso un determinato canale (dispositivo di output).
È necessario eseguirla per poter indirizzare correttamente tutte le operazioni di *input/output* che seguiranno.

reg. in:
A = codice del flusso

reg. out:
IX non modificato
altri modificati

Nota: codici flusso previsti:

0, 1, 253	canale "K" (screen edit area, righe 23-24) equivalente a OPEN 1, "K"
2, 254	canale "S" (screen display area, righe 0 ÷ 21) equivalente OPEN 2, "S"
3	canale "P" (printer) equivalente OPEN 3 "P"
255	canale "R" (area di memoria indirizzata dalla var.

KCUR, normalmente la edit-area), non consentito in BASIC

A conclusione dei previsti controlli, imposta le variabili:
FLAGS (23611) con bit 1 ON se "P"
TIFLAG (23612) con bit 0 ON se "K", OFF se "S"

PPOS-SET (CALL dec 3545 hex 0DD9)

Imposta al valore voluto la *current screen/print position:* a seconda del canale precedentemente selezionato (S-K-P), le variabili interessate (da PPOSN a SPOSNL) vengono impostate con la posizione corrente di stampa, le cui coordinate sono contenute in BC.

reg. in:
B = coord. riga = 24 - num. riga = da 24 a 3 se "S"; da 24 a 23 se "K"; ignorato se "P"
C = coord. colonna = 33 - num. colonna = da 33 a 2

reg. out:
HL = corrispondente indirizzo *row 0* in display-file o printer-buffer
A e DE = num. colonna risultante (0-31)
BC e IX non modificati

PRINT (RST dec 16 hex 10)

Un'unica routine gestisce tutti i possibili tipi di output, trattando contemporaneamente attributi e codici di controllo.

Essendo molto utilizzata, si trova in pagina 0 e viene invocata con l'istruzione di restart RST 16 di un solo byte (equivalente a CALL 16 di 3 bytes).

reg. in:
A = CODE del carattere o del codice
reg. out:
A modificato
altri non modificati

Note: la stampa avviene alla *current screen / print position* indicata dalle variabili DFSZ - PPOSN - PRCC - ECHOE - DFCC - DFCCL - SPOSN - SPOSNL, che vengono poi aggiornate, utilizzando il canale precedentemente selezionato.

Attenzione: con canale "P" la locazione 23681, dichiarata non usata, viene invece impostata a dec. 91 (byte alto dell'indir. del *printer buffer*).

Ai dati stampati vengono associati gli attributi temporanei, in base al contenuto delle variabili ATTRT - MASKT - BORDCR - PFLAG; gli attributi temporanei, se non specificatamente impostati, contengono la copia di quelli permanenti.

Le prime tre hanno il seguente formato:
bits 0÷2 = valore INK 0÷7
bits 3÷5 = valore PAPER (0÷7) * 8
bit 6 ON = BRIGHT attivo
bit 7 ON = FLASH attivo

P FLAG ha il seguente formato (in cui i bits pari 0-2-4-6 indicano gli attr. temporanei e i bits dispari 1-3-5-7 quelli permanenti):
bits 0/1 ON = OVER attivo
bits 2/3 ON = INVERSE attivo
bits 4/5 ON = INK 9
bits 6/7 ON = PAPER 9

Come detto, la capacità di trattare i codici di controllo consente di passarli serialmente alla routine, tramite il reg. A; per esempio i valori decimali:
16,2 equivalgono a INK 2
20,1 equivalgono a INVERSE 1
22,12,5 equivalgono a AT 12,5

Utilizzando la routine PRINT-STRING che segue, i codici di controllo possono essere inclusi nella stringa.

TEMP-ATT (CALL dec. 3405 hex 0D4D)
copia gli attributi permanenti in quelli temporanei.
reg. out:

A e HL modificati
altri non modificati

PERM-ATT (CALL dec 7341 hex 1CAD)
Copia gli attributi temporanei in quelli permanenti
reg. out:
A e HL modificati
altri non modificati

PRINT-STRING (CALL dec 8252 hex 203C)
stampa una stringa di caratteri.
reg. in:
DE = indirizzo della stringa
BC = lunghezza della stringa
reg. out:
DE = indirizzo fine stringa + 1
A = 0
BC = 65535
IX e HL non modificati

PRINT-NUM1 (CALL dec 6683 hex 1A1B)
Converte un numero binario in numero decimale, e lo stampa allineato a sinistra, sopprimendo gli zeri non significativi.
reg. in:
BC = valore binario
reg. out:
A e BC modificati
altri non modificati
Nota: funziona correttamente solo con valori da 0 a 9999.

PRINT-NUM 2 (CALL dec 6696 hex 1A28)
converte un numero binario in numero decimale, e lo stampa allineato a destra, sostituendo con spazi gli zeri non significativi.

reg. in:
HL = indir. area di due bytes, contenente il valore binario nella forma bytes alto + basso (come un line-number basic).
reg. out:
HL incrementato di 1
A e BC modificati
IX e DE non modificati
Nota: funziona correttamente solo con valori da 0 a 9999.

PRINTNUM: entrambe le routines possono essere utilizzate per ottenere risultati migliori.
Si può usare questa subroutine invocandola con CALL PRNUM:

```
9C40               5          ORG  40000
                   6
                   7
9C40 D5           10 prnum    PUSH DE
9C41 E5           20          PUSH HL
9C42 011027       30          LD   BC,10000
     ;non necessario se num.max 9999
9C45 CD2A19       40          CALL 6442
     ;idem
9C48 C3301A       50          JP   6704
9C4B              60          END
```

reg. in:
HL = valore binario da 0 a 65535 (= 5 cifre decimali)
E = flag di edit
reg. out:
A e BC modificati
altri non modificati
Nota: il flag da caricare nel reg. E consente di ottenere risultati diversi
255 = numero allineato a sinistra, con soppressione degli

zeri non significativi;

32 = numero allineato a destra, con zeri non significativi sostituiti da spazi;

48 = numero completo di zeri non significativi.

33 ÷ 127 = num. allineato a destra; gli zeri non significativi vengono sostituiti con il carattere corrispondente al CODE di flag; es.: flag 96 = &.&. 123

PRINT-FP (CALL dec 13236 e 11747 hex 33B4 e 2DE3)

La prima routine (STACK - FP) può essere necessaria per caricare il *calculator stack* con il numero *floating point* contenuto in un'area di 5 bytes; la seconda converte il *last-value* del calc. stack in decimale e lo stampa.

reg. in:

HL = indirizzo area contenente il numero f.p.

reg. out.:

A e BC = 0

IX non modificato

HL e DE modificati

Nota: il calc. stack e le routines che lo gestiscono sono specificamente trattati al capitolo *Il calcolatore f.p.*

PRINT-LINE 1 (CALL dec 6245 hex 1865)

Stampa una linea basic, compreso il numero di linea.

reg. in:

HL = indirizzo area contenente la linea basic, normalmente nell'area basic o in quella di edit, ma può essere una qualsiasi locazione di RAM; HL deve puntare il byte alto del numero di linea.

D = 0, se non occorre print del cursore, oppure CODE del simbolo rappresentante il cursore (normalmente dec. 62)

reg. out:

HL = indirizzo inizio linea basic successiva

A = 13

BC modificato

IX e DE non modificati

Nota: nel caso in cui la routine venga utilizzata ricorsivamente per listare una sequenza di linee BASIC, può rendersi necessario prevenire un rientro scorretto: infatti, verificandosi il superamento dell'area basic, la routine scarica dallo stack l'indirizzo di ritorno e rientra al livello superiore. Per tale evenienza, prima di invocare la routine, può essere necessario caricare lo stack con l'indirizzo a cui rientrare dopo l'elaborazione dell'ultima linea basic.

PRINT-LINE 2 (CALL dec 6269 hex 187D)
stampa una linea basic escludendo il numero di linea.
reg. in:
HL = come PRINT - LINE 1, ma il byte puntato è quello, all'interno del testo della linea, da cui si vuole iniziare la stampa.
reg. out:
BC non modificato

altri come PRINT-LINE 1
Nota: al rientro da entrambe le routines PRINT-LINE, se necessita un ritorno a capo, eseguire RST 16.

PRINT - E/I (CALL dec. 4381 hex 111D)
Stampa nella parte bassa dello schermo il contenuto della *edit-area* (ELINE) o della *work - space - area* (WORKSP), in funzione del bit 5 della var. FLAGX (23665): ON = WORKSP, OFF = ELINE.
L'area da stampare deve terminare col codice NL (dec. 13, newline).
reg. out:
A e DE = 0
BC e HL modificati
IX non modificato
Nota: prima di chiamare questa routine, deve essere aperto

il can. "K" (LD A, 1: CALL 5633), e la variabile KCUR (anche solo il byte alto) azzerata (LD (IY + 34), 0).

PRINT-MESS (CALL dec 3082 hex 0C0A)

Stampa un messaggio, o una stringa, ricercandolo in una tabella di elementi a lunghezza variabile.

reg. in:
A = numero dell'elemento cercato (0 = primo el.)
DE = indirizzo inizio tabella
reg. out:
IX e HL non modificati
altri modificati

Nota: il primo byte della tabella essere un qualsiasi CODE maggiore o uguale a 128 (viene by-passato); l'ultimo byte di ciascun elemento, per essere riconosciuto, deve avere il bit 7 ON (= CODE del carattere + 128).

È questa la routine e il modo con cui il sistema gestisce e stampa tokens e reports.

PRINT-ERR (RST dec 8 hex 08)

Anche questa è una routine in pagina 0, eseguibile con RST 8; segnala nella *screen - edit - area* il messaggio di errore *(report)*, e termina immediatamente l'esecuzione del programma, ritornando in modo *comandi*.

L'istruzione deve essere seguita da un byte indicante il codice di errore — 1; esempio:
255 = 0 OK
13 = E out of data

LC-SET (CALL dec 3545 hex 0DD9)

A seconda del canale aperto, calcola la posizione di stampa in base al contenuto dei reg. B e C, e imposta variabili le PPOSN - PRCC (se "P"), SPOSN - DFCC (se "S"), SPOSNL -ECHOE - DFCCL (se "K").

reg. in:
B = 24 - num. riga (num. riga = 0 ÷ 21 se "S", 0 ÷ 1 se "K") ed è ignorato se "P"
C = 33 - num. colonna (num. col. = 0 ÷ 31)
reg. out:
IX e BC non modificati
altri modificati.

LC-ADDR (CALL dec 3739 hex 0E9B)
Calcola l'indirizzo di display-file corrispondente a *colonna 0, row 1* di una determinata riga.
reg. in:
B = 24 - num. riga (num. riga = 0 ÷ 23)
reg. out:
HL = indirizzo corrispondente di *display-file*
A e D modificati
altri non modificati

CLS (CALL dec 3503 hex 0DAF)
Esegue un completo *clear-screen*, resettando al valore iniziale le variabili da COORDS a SCRCT.
reg. out:
B = 24 (24-riga 0)
C = 33 (33-colonna 0)
HL = indirizzo *display-file* = 16384
A e DE = 0
IX non modificato

CLS-BORD (CALL dec 3435 hex 0D6B)
come CLS; in più assegna alle righe della *screen-edit-area* gli attributi di colore presenti nella variabile BORDCR.
reg. out:
B = 23 (24 - riga 1)
C = 33 (33 - colonna 0)
HL = indirizzo *display-file* riga 23 col. 0

A e DE = 0
IX non modificato

CLS-LOW (CALL dec 3652 hex 0E44)
Esegue il CLS delle ultime *n* righe dello schermo, lasciando inalterate le variabili da COORDS a SCRCT, e quindi anche le *current screen positions*
reg. in:
B = *n* (numero di righe da cancellare = 1 ÷ 24)
reg. out:
C = 33
B e IX non modificati
altri modificati

CLS-ED (CALL dec 3438 hex 0D6E)
Esegue il CLS della sola *screen edit area*, usando gli attributi presenti in BORDCR; le var. DFSZ - SPOSNL - ECHOE -DFCCL vengono resettate ai valori iniziali.
reg. out:
B = 23 (24 - riga 1)
C = 33 (33 - colonna 0)
HL = indirizzo *display-file* riga 23 col. 0
A e DE = 0
IX non modificato

SCROLL-1 (CALL dec 3582 hex 0DFE)
Esegue lo *scroll-up* di una riga, estendendolo all'intero schermo: riga 1 in 0, 2 in 1, ..., 23 in 22; poi esegue il *clear* di riga 23.
La *current screen position* non viene modificata.
reg. out:
B = 1 (24 - riga 23)
C = 33 (33 - colonna 0)
IX non modificato
altri modificati

SCROLL-2 (CALL dec 3584 hex 0E00)

Consente di effettuare lo scroll a partire da una determinata riga. La *current screen position* non viene modificata.

Es.: con reg. B = 15 (24-15 = riga 9), si ottiene lo scroll di riga 9 in 8, 10 in 9, ..., 23 in 22, e infine il *clear* di riga 23.

reg. in:
B = 24 - numero della prima riga da "scrollare" (1 ÷ 23)
reg. out:
come la routine SCROLL-1.

PLOT (CALL dec 8933 hex 22E5)

Esegue l'analogo comando basic, impostando contemporaneamente la variabile COORDS (23677)

reg. in:
B = coordinata Y (0 ÷ 175)
C = coordinata X (0 ÷ 255)
reg. out:
IX non modificato
altri modificati

Nota: la routine gestisce anche le funzioni di OVER e INVERSE, testando la var. PFLAG (23697):
bit 0 ON = OVER 1
bit 2 ON = INVERSE 1

DRAW (CALL dec 9402 hex 24BA)

Esegue l'analogo comando basic, aggiornando la variabile COORDS (23677).

reg. in:
B = valore assoluto di Y (spostamento verticale)
C = valore assoluto di X (spostamento orizzontale)
D = segno di Y
E = segno di X
(1 = posit., 255 = negat.)
reg. out:
IX e DE non modificati
altri modificati

Nota: se necessita poter rientrare al BASIC, il reg. HL' deve essere salvato (EXX: PUSH HL : EXX), prima di CALL 9402, e poi ripristinato (EXX: POP HL : EXX).

PIX - ADDR (CALL dec 8874 hex 22AA)

Calcola l'indirizzo in *display-file* del pixel, le cui coordinate sono date in BC.

reg. in:
B = coordin. Y (0 ÷ 175)
C = coordin. X (0 ÷ 255)
reg. out:
HL = indir. byte in *display - file*
A = posizione del pixel nel byte (0 ÷ 7)
BC modificato
IX e DE non modificati

POINT (CALL dec 8910 e 11733 hex 22CE e 2DD5)

La prima routine esegue l'analoga funzione basic, caricando il risultato nel *calc. stack*; la seconda trasferisce il valore nel reg. A.

reg. in:
B = coordinata di Y (0 ÷ 175)
C = coordinata di X (0 ÷ 255)
reg. out:
A = risultato della funzione (1 o 0)
BC = A
IX non modificato
altri modificati

ATTR (CALL dec 9603 e 11733 hex 2583 e 2DD5)

La prima routine esegue l'analoga funzione basic, caricando il risultato nel *calc. stack*; la seconda trasferisce il valore nel reg. A.

reg. in:
B = numero della colonna

C = numero della riga
reg. out:
A = risultato della funzione (0 ÷ 255)
BC = A
IX = non modificato
altri modificati

SCREEN $ (CALL dec 9528 e 11249 hex 2538 e 2BF1)

La prima routine esegue l'analoga funzione basic, caricando il risultato nel *calc. stack*; la seconda trasferisce il valore nel reg. A.

reg. in:
B = num. colonna
C = num. riga
reg. out:
A = risultato della funzione (32 ÷ 127)
IX non modificato
altri modificati

L'area BASIC

L'area basic

LINE-COMP (CALL dec 6528 hex 1980)

Compara il numero di linea contenuto in BC con quello contenuto nei due bytes puntati da HL; ricordare che, trattandosi di linee basic, questi due bytes sono invertiti: alto + basso.

reg. in:
BC = numero di linea cercato
HL = indirizzo linea (in area basic)
reg. out:
A modificato
altri non modificati
flags out:
Z/NZ = numero di linea uguale / diverso
C = numero di linea in area basic minore del numero di linea cercato.

LINE-SEARCH (CALL dec 6510 hex 196E)

Cerca un numero di linea o quello immediatamente successivo; la ricerca parte sempre dall'inizio dell'area basic.

reg. in:
HL = numero di linea cercato
reg. out:
BC = numero di linea cercato
A = modificato
IX non modificato
HL e DE modificati secondo le risultanze
flags out:
Z = trovato numero di linea cercato:
HL = indirizzo linea trovata
DE = indirizzo linea precedente (o inizio area basic se la linea trovata è la prima)
NZ e (HL) minore di 64 = trovato numero di linea successivo:

HL = indirizzo linea trovata
DE = indirizzo linea precedente (o inizio area basic se la linea trovata è la prima)
NZ e (HL) maggiore o uguale a 64 = numero di linea non trovato:
HL = indirizzo inizio area variabili
DE = indirizzo ultima linea basic

LINE - NEXT (CALL dec 6584 hex 19B8)
Calcola l'indirizzo della linea basic o della variabile successiva, a seconda di quale area è indirizzata da HL.
reg. in:
HL = indirizzo di una linea basic o di una variabile
reg. out:
DE = indirizzo next-line o next-variable
BC = lunghezza linea o variabile (= DE - HL)
A = 0
IX e HL non modificati

Nota:
a) trattando linee basic, se (DE) maggiore o uguale a 64, significa che è esaurita l'area basic e
HL = indirizzo inizio ultima linea basic
DE = indirizzo inizio area variabili
b) trattando variabili, se (DE) = 128 significa che è esaurita l'area delle variabili e
HL = indirizzo ultima variabile
DE = indirizzo edit-area —1

FP-SKIP (CALL dec 6326 hex 18B6)
By-passa la rappresentazione *floating-point* di un numero, se reg. A = 14 *(f.p. control character).*
reg. in:
HL = indirizzo area in esame
A = (HL) = contenuto del byte puntato da HL

reg. out:
IX, BC e DE non modificati
A e HL non modificati se A diverso da 14, altrimenti
HL = indirizzo iniziale + 6
A = (HL) = contenuto nuova locazione puntata da HL

La memoria

La memoria

FREE-SPACE (CALL dec 7962 hex 1F1A)

calcola l'ampiezza dello *spare space*, corrispondente alla quantità di memoria disponibile, esistente tra STKEND e SP (inizio area *machine stack*).

reg. out:
BC = numero (complementato a 65536) di bytes liberi
A e IX non modificati
DE e HL modificati

Nota: in ogni momento è possibile conoscere la quantità di memoria disponibile eseguendo PRINT 65536 - USR 7962.

SPACE-CALC (CALL dec 6621 hex 19DD)

calcola l'ampiezza di un'area per differenza tra due indirizzi X - Y.

reg. in:
HL = indirizzo X
DE = indirizzo Y
reg. out:
DE e HL invertiti
BC = differenza calcolata +/—
A e IX non modificati

SPACE-ACQ 1 (CALL dec 5717 hex 1655)

acquisisce uno spazio in memoria, pari al valore di BC, shiftando conseguentemente le aree di sistema coinvolte, e aggiornando del pari le variabili che le indirizzano.

```
prima   X Y                dopo    X   spazio       Y
-----------|---------      --------|--------------|--------
         12|345...               12|  acquisito   |345...
```

reg. in:
HL = indirizzo y da cui inizia lo shift
BC = numero di bytes da acquisire
reg. out:
HL = indirizzo x
DE = indirizzo y1 - 1 se y è minore di STKBOT
oppure x se y è maggiore o uguale a STKBOT
BC = 0
A e IX non modificati

Nota: lo shift avviene solo dopo aver testato che esista in *spare space* lo spazio richiesto + ulteriori 80 bytes (CALL 7941), altrimenti si ha errore di "Out of memory".

Questa routine, e le altre tre che seguono, vengono normalmente utilizzate per acquisire o rilasciare spazio nelle aree di sistema, da PROG a STKBOT.

SPACE-ACQ2 (CALL dec 5722 hex 165A)

identica alla precedente, senza esecuzione del test di memoria disponibile.

SPACE-REL1 (CALL dec 6629 hex 19E5)

rilascia uno spazio di memoria, shiftando conseguentemente le aree di sistema coinvolte, e aggiornando del pari le variabili che le indirizzano.

```
prima X spazio       Y             dopo     X Y
______|-------------|______      ___________|_________
   12| da rilasc. |345...               12|345...
```

reg. in:
DE = indir. x
HL = indir. y (con y maggiore di x)
reg. out:
HL = indir. x

BC = 0
A e DE modificati
IX non modificato

SPACE-REL 2 (CALL dec 6632 hex 19E8)
identici risultati della precedente
reg. in:
HL = indirizzo di x
BC = numero dei bytes da rilasciare
reg. out:
BC = 0
IX e HL non modificati
A e DE modificati

WORKSP - ACQ (RST dec 48 hex 30)
routine in pagina 0; alloca spazio nella *work-space area*, accodandolo a quello eventualmente già esistente.
reg. in:
BC = numero di bytes da acquisire
reg. out:
HL = indirizzo fine spazio aggiunto (STKBOT - 1)
DE = indirizzo inizio spazio aggiunto
A, BC e IX non modificati

RAM-LOAD (CALL dec 3969 hex 0F81)
consente di caricare dati in memoria, serialmente un byte dopo l'altro, all'indirizzo indicato dalla variabile KCUR (23643): questa viene via via incrementata di 1.
L'area di memoria da caricare, normalmente *edit-area* o *work-space*, viene acquisita automaticamente, un byte alla volta, tramite la routine SPACE-ACQ 1.
reg. in:
A = CODE del dato da caricare
reg. out:
DE = indirizzo di memoria in cui è stato caricato A (attuale

KCUR - 1)
HL = DE - 2
BC = 0
A e IX non modificati

Nota: è curioso annotare che lo stesso risultato si può ottenere utilizzando l'istruzione di print (RST 16), dopo aver aperto lo speciale canale "R" (LD A, 255 : CALL 5633).

CLEAR - EICS (CALL dec 5808 hex 16B0)
esegue il clear di *edit area, work-space e calculator stack,* da ELINE e STKEND, rilasciandone tutto lo spazio.
reg. out:
HL = STKEND = inizio *spare-space*
altri non modificati

CLEAR-ICS (CALL dec 5823 hex 16BF)
esegue il clear di *work-space* e *calculator stack*, da WORKSP e STKEND, rilasciandone lo spazio.
reg. out:
come CLEAR-EICS

CLEAR - CS (CALL dec 5829 hex 16C5)
esegue il clear del *calculator stack*, rilasciandone lo spazio.
reg. out: come CLEAR-EICS

CLEAR - E/I (CALL dec 4247 hex 1097)
esegue il clear, rilasciandone lo spazio, di *edit-area* o *work-space*, in funzionamento del bit 5 della variabile FLAGX (23665) : ON = WORKSP, OFF = ELINE.
Al termine, la variabile KCUR (23643) è impostata con l'indirizzo di inizio dell'area trattata.
reg. out:

BC = 0
IX = e HL non modificati
A e DE modificati

CLEAR (CALL dec 7855 hex 1EAF)
esegue l'analogo comando basic.
Modificando RAMTOP, il *machine stack* viene riallocato al di sotto del nuovo limite, e deve essere ricostituito, almeno con i due indirizzi di ritorno necessari a interprete ed esecutore dei comandi.
reg. in:
BC = nuovo valore di RAMTOP, oppure 0 se RAMTOP non deve essere modificato.
reg. out:
BC = *interpr. return,* contenuto nella prima entrata dello stack (indirizzo di ritorno da RAND USR *n*)
HL = *return address* (indirizzo di ritorno da CALL 7855)
DE = RAMTOP-1
A = 0
IX = non modificato
Nota: per utilizzare la routine e ricostituire correttamente lo stack, occorre eseguire le seguenti istruzioni:
LD HL, 4867
EX (SP), HL
CALL 7855
LD BC, 7030
PUSH BC
a questo punto lo stack contiene *(bottom to top)*: *error return, exec. return* (4867), *interpr. return* (7030).

Il calcolatore in virgola mobile

Il calcolatore floating point

Importantissima componente del sistema, il calcolatore floating point è un vero e proprio computer nel computer, e viene utilizzato per trattare valori numerici e stringhe.

Come area principale di lavoro, utilizza il *calculator stack*: questo è molto simile al *machine stack*, e analogamente gestito con tecnica LIFO *(last in - first out)*, ma ogni valore immessovi si compone di 5 bytes.

Le var. STKBOT (23651) e STKEND (23653) ne indirizzano rispettivamente l'inizio e la fine + 1.

Col termine *last value* (di seguito abbreviato in LV) viene indicato il valore contenuto in cima allo stack, cioè l'ultimo valore caricato.

Viene invocato con l'istruzione di restart RST dec. 40 (hex 28), seguita da un numero variabile di bytes, contenenti i codici delle operazioni da eseguire (*literals*); l'ultimo codice deve essere un dec 56, a indicare *end-calc*.

Normalmente, ogni singola operazione scarica dallo stack i valori utilizzati, e registra un nuovo LV, contenente il risultato.

Alcuni codici sono, per esempio:

3 *(sub)*:	LV = LV — 1 — LV (usando la formula a-b = a + (-b))
15 *(add)*:	LV = LV-1 + LV
4 *(mult)*:	LV = LV-1 * LV
5 *(div)*:	LV = LV-1 / LV
1 *(exchange)*:	LV = LV-1 e LV-1 = LV
160 *(stk - 0)*:	inserisce un nuovo LV = 0
161 *(stk - 1)*:	inserisce un nuovo LV = 1
162 *(stk - 1/2)*:	inserisce un nuovo LV = 1/2
163 *(stk - PI/2)*:	inserisce un nuovo LV = PI/2
164 *(stk - 10)*:	inserisce un nuovo LV = 10
2 *(del)*:	annulla il LV decrementando di un valore (5 bytes) il puntatore.
49 *(dupl)*:	inserisce un nuovo LV = LV-1

50 *(to-power)*: LV = LV-1 ↑ LV
40 *(sqr)*: LV = SQR (LV)

Molte sono le funzioni ottenibili (jump, logiche AND - OR -NOT, VAL, CHR$, LEN, SIN, TAN, etc..., comparazioni), consentendo di svolgere completi sottoprogrammi ed equazioni anche complesse, concatenando singole operazioni.

Per esempio, avendo in LV un numero indicante il raggio di un cerchio, eseguendo

RST 40
DEFBs 49, 4, 163, 4, 49, 15, 56
si ottiene come nuovo LV la superficie.

49 = *dupl* =	R, R
4 = *mult* =	R * R
163 = *stk-PI/2* =	R * R, PI/2
4 = *mult* =	R * R * PI/2
49 = *dupl* =	R * R * PI/2, R * R * PI/2
15 = *add* =	R * R * PI
56 = *end - calc*	

STACK-A (CALL dec 11560 hex 2D28)

registra nel *calculator stack* il numero *floating point* corrispondente al valore binario assoluto contenuto in A.

reg. in:
A = valore binario da trattare
reg. out:
DE = STKEND (precedente + 5)
HL = STKEND-5
A e BC modificati
IX non modificato

STACK-DIGIT (CALL dec 11554 hex 2D22)

registra nel calculator stack il numero *floating point* corrispondente al numero ASCII contenuto in A; se A è compreso

tra 48 e 57, lo stack non viene caricato e si ha un ritorno con flag. C.

reg. in:
A = CODE ASCII da trattare
reg. out:
DE = STKEND (precedente +5)
HL = STKEND — 5
A e BC modificati
IX non modificato
flags out:
C se A non contiene un numero

STACK-BC (CALL dec 11563 hex 2D2B)
registra nel *calculator stack* il numero *floating point* corrispondente al volore binario assoluto contenuto in BC.

reg. in:
BC = valore binario da trattare
DE = STKEND (precedente + 5)
HL = STKEND-5
A e BC modificati
IX non modificato

STACK-INT (CALL dec 11579 hex 2D3B)
converte un numero decimale intero in formato *floating point* e lo registra nel *calculator stack;* l'area di memoria contenente il numero decimale è indirizzata dalla variabile CHADD (23645), e la fine del numero è interpretata da un digit non numerico (compreso tra 48 e 57).

reg. out:
DE = STKEND (precedente + 5)
HL = indirizzo fine area (= digit non numer.)
A e BC modificati
IX non modificato

Note: prima di CALL 11579, è necessario eseguire un RST 24 per leggere il primo digit. Il numero floating point ottenuto è sempre assoluto (positivo): può essere trasformato in negativo, utilizzando la routine LV-NEG.

Se necessita poter rientrare al BASIC, la variabile CHADD deve essere reimpostata, in modo da puntare a una locazione contenente un codice NL dec. 13 (per esempio WORKSP-2), oppure prima salvata e poi ripristinata.

STACK-DEC (CALL dec 11448 hex 2CB8)

Converte un numero decimale, anche esponenziale, in formato *floating point*, e lo registra nel *calculator stack*; come nella routine precedente, l'area contenente il numero decimale è indirizzata dalla variabile CHADD (23645), mentre la fine del numero è interpretata da un codice non numerico (non compreso tra 48 e 57, e diverso da 46-69-101).

Es. di numeri decimali validi:
235 - 2.35 - .23 - 2.35 E 1 - 2.35 E - 1 - 2 E 3.

reg. out:
DE = STKEND (precedente + 5)
IX = non modificato
altri modificati
Note: vedi STACK-INT

LV-NEG (CALL dec 13422 hex 346E)
LV-ABS (CALL dec 13418 hex 346A)

consentono di modificare il segno del LV nel *calculator stack*:

LV-NEG inverte il segno
LV-ABS elimina il segno ottenendo un valore assoluto
reg in:
HL = STKEND-5 = primo byte di LV
reg. out:
A e BC modificati
altri non modificati

HL & DE - SET (CALL dec 13759 hex 35BF)
imposta i reg. HL e DE con gli indirizzi rispettivamente di STKEND-5 (LV) e STKEND.
reg. out:
HL e DE impostati
altri non modificati

Nota: lo stesso risultato si può anche ottenere con RST 40 : DEFB 56;

STACK-TO-BC (CALL dec 11682 hex 2DA2)
il LV nel *calculator stack* è arrotondato all'intero e convertito in un valore assoluto binario.
reg. out:
BC = risultato (valido se minore di 65535)
A = C
DE = STKEND (precedente - 5)
HL = STKEND - 5
IX non modificato
flags out:
C = valore assoluto *floating point* originario maggiore di 65535
NZ = valore originario negativo

STACK-TO A (CALL dec 11733 hex 2DD5)
stesso risultato della precedente routine, che viene utilizzata per arrotondamento e conversione.
reg. out:
A = risultato (valido se minore o uguale a 255)
C = A
DE = STKEND (precedente -5)
HL = STKEND - 5
B = modificato
IX non modificato
flags out:
C = valore assoluto *floating point* originario maggiore di 255

NZ = valore originario negativo

STACK-STORE (CALL dec 10934 hex 2AB6)
memorizza nel *calculator stack* il contenuto dei reg. A - E -D - C - B.
reg. in:
A - E - D - C - B da caricare nello stack
reg. out:
HL = STKEND (precedente + 5)
altri non modificati

STACK-FETCH (CALL dec 11249 hex 2BF1)
scarica il LV dal *calculator stack* e lo trasferisce nei reg. A-E-D-C-B
reg. out:
A - E - D - C - B contenenti il LV
HL = STKEND (precedente - 5)
IX = non modificato

STACK-FP (CALL dec 13236 hex 33B4)
carica nel *calculator stack* il numero *floating point* contenuto in un'area di 5 bytes.
reg. in:
HL = indirizzo area contenente il num. *floating point*.
reg. out:
HL = indirizzo area + 5
DE = STKEND (precedente + 5)
BC = 0
A, DE e IX non modificati

VAR-SEARCH (CALL dec 10418 hex 28B2)
ricerca una variabile o un array nell'area delle variabili. Il nome della variabile / array cercati è contenuto in un'area di memoria indirizzata dalla variabile CHADD, ed è composto come avviene in una linea basic; la fine del nome è interpreta-

to da un byte non contenente nè un digit numerico, nè una lettera (per es. ".", ma per arrays numerici usare "(").

variab. in

DEFADD (byte alto di 23564) deve essere = 0

FLAGS (23611) bit 7 on

reg. out:

C = tipo di variabile:

bit 5 OFF e 6 OFF = array numerico

bit 5 OFF e 6 ON = variabile o array stringa

bit 5 ON e 6 ON = variabile o array numerico a nome corto (1 carattere)

bit 5 ON e 6 OFF = variabile numerica a nome lungo.

A e B modificati

IX non modificato

DE e HL = secondo le risultanze

flags out:

C = variabile non trovata:

HL = inizio nome variabile in area di memoria

DE = HL + 1

NC = variabile trovata:

HL = indir. ultimo o unico carattere del nome in area variabili

DE = inizio nome var. in area di mem.

Nota: se necessita poter rientrare al basic, la variabile CHADD deve essere reimpostata, in modo da puntare a una locazione contenente un cod. NL dec 13 (per es. WORKSP-2), oppure prima salvata e poi ripristinata.

VAR-STACK (CALL dec 10646 hex 2996)

carica nel *calculator stack* i parametri (indirizzo e lunghezza) di una variabile stringa, o di un determinato elemento di un array, oppure uno *slice* di queste.

La routine è abbastanza complessa, ed è opportuno chiamarla immediatamente dopo la precedente VAR-SEARCH, di cui utilizza registri e flags risultanti.

Se è richiesto un preciso elemento di un array e/o uno *slice*, l'area di memoria indirizzata da CHADD deve contenere anche tali informazioni; es.: X $ (2). oppure X $ (2 TO 4), X $ (2,3 TO 5), etc. o Y (5, 1, 3), etc.

Attenzione: i dati numerici di elemento e/o *slice* debbono essere seguiti dalla rappresentazione *floating point*, così come avviene in una linea basic; es.: X $ (2) = dec. 120 36 40 50 14 0 0 2 0 0 41.

reg. out.:

DE = indirizzo dei dati della variabile o elemento o *slice* in area variabili

BC = lunghezza dei dati richiesti

HL = indirizzo fine area memoria + 1 (= carattere non numerico)

A modificato

IX non modificato

Nota: i valori risultanti in DE e BC sono stati anche caricati nel *calculator stack*, e da lì possono essere acquisiti utilizzando, per esempio, la routine STACK-FETCH. In ogni caso, è opportuno scaricarli dal *calculator stack,* se non necessari.

ADD (CALL dec 12308 hex 3014)
SUB (CALL dec 12303 hex 300F)
MULT (CALL dec 12490 hex 30CA)
DIV (CALL dec 12719 hex 31AF)

sono le routines usate dal calcolatore per eseguire le corrispondenti operazioni in virgola mobile, tutte agiscono sugli operandi esistenti in LV-1 e LV, restituendo nel LV finale il risultato.

reg. in:
HL = indir. LV-1 = STKEND - 10
DE = indir. LV = STKEND - 5
reg. out:
HL = indir. LV = STKEND - 5
DE = STKEND
A e BC modificati

IX non modificato
Nota: gli stessi risultati si ottengono, rispettivamente, con:
RST 40: DEFBs 15,56
RST 40: DEFBs 3,56
RST 40: DEFBs 4,56
RST 40: DEFBs 5,56
cosí facendo, gli indirizzi iniziali in HL e DE vengono impostati automaticamente.

HL * DE (CALL dec 12457 hex 30A9)
esegue la moltiplicazione dei valori in HL e DE.
reg. in:
DE e HL = moltiplicando e moltiplicatore
reg. out:
HL = prodotto
A modificato
altri non modificati
flags out:
C = prodotto maggiore di 65535

L'editor

ED-INPUT (CALL dec 3884 hex 0F2C)

La routine governa la costruzione e l'edit di linee basic nell'*edit-area* (ELINE) e l'immissione di dati nell'*input-area* (WORKSP): quest'ultima condizione è quella che offre le maggiori possibilità di uso pratico.

Una volta chiamata, la routine entra in un loop di gestione dell'input da tastiera, che termina solo dopo la pressione di ENTER.

Usata come INPUT, richiede la variabile FLAGX (23665) con bit 5 ON (a indicare uso dell'*input-area*), e la variabile KCUR (23643) contenente l'indirizzo di inizio dell'area (WORKSP): in questa locazione deve trovarsi già memorizzato un codice NL (dec. 13).

KCUR viene incrementata e, contemporaneamente, l'*input-area* ampliata di un byte per ogni carattere immesso.

reg. out: tutti modificati

Note: prima di chiamarla è necessario aprire il canale K con LD A, 1 : CALL 5633.

In uscita, le variabili WORKSP (23649) e STKBOT (23651) indicano rispettivamente l'inizio dell'area acquisita e la fine + 1: l'ultimo carattere immesso è sempre seguito dal codice NL.

Si può provare la seguente routine:

```
9C40                5        ORG  40000
9C40 FDCB37EE      10        SET  5,(IY+55)
     ;FLAGX bit 5 on
9C44 CDBF16        20        CALL 5823
     ;clear-ICS
9C47 010100        30        LD   BC,1
9C4A F7            40        RST  48
     ;acquisisce 1 byte in WORKSP
9C4B 360D          50        LD   (HL),13
     ;vi carica il cod.NL
```

```
9C4D 225B5C        60          LD   (23643),HL
     ;KCUR=indirizzo WORKSP
9C50 3E01          70          LD   A,1
9C52 CD0116        80          CALL 5633
     ;open can."k"
9C55 CD2C0F        90          CALL 3884
     ;ED-INPUT
                  100 ;rientrando qui, l'input-
     area contiene i dati immessi
```

INPUT (CALL dec 8329 hex 2089)

esegue l'analogo comando basic.

La variabile CHADD (23645) deve indirizzare l'area di memoria contenente i dati dello statement INPUT (escluso il comando): l'area deve terminare con un codice di *end - statement* (per es. NL dec. 13)

La var. FLAGS (23611) deve avere il bit 7 ON, a indicare *run*, anzichè *syntax-check*. L'eventuale variabile coinvolta nello statement viene inserita o riassegnata nell'area delle variabili; i dati immessi sono anche disponibili in *input-area* (WORKSP), seguiti da un codice NL dec. 13, nel seguente formato:

se var. stringa: *"(stringa)"*

se var. stringa con LINE: *stringa*

se var. numerica: *numero decimale ASCII*, seguito dal codice decimale 14 e dalla rappresentazione in virgola mobile.

reg. out:

A e DE = 0

altri modificati

Nota: in uscita, le variabili WORKSP (23649) e STKBOT (23651) indicano rispettivamente l'inizio della *input-area* acquista e la fine + 1.

Esempio di routine chiamante:

```
9C40               10          ORG  40000
9C40 FDCB01FE      20          SET  7,(IY+1)
```

```
     ;FLAGS bit 7 on
9C44 22449C       30         LD   (CHADD),HL
     ;indir.area cont.per es."Nome? ";n$

9C47 CD8920       40         CALL 8329
     ;input
9C4A C9           50         RET
```

Al rientro, la variabile *n$* si trova impostata nell'area variabili, e l'*input-area* contiene la stringa immessa, seguita dal codice NL.

Attenzione: l'area di memoria indirizzata da CHADD deve trovarsi a indirizzi più bassi dell'area variabili: quindi nel *printer-buffer* o nell'area basic, resa disponibile con REM, oppure nell'area MEMBOT (23698), se per uso temporaneo.

Non usando i Microdrives, un modo interessante per disporre di un'area di lavoro, utile anche in altre occasioni, consiste nell'allocare una *microdrives maps area* all'indir. 23734, shiftando di quanto occorre la *channel data area* e le aree seguenti; si può ottenere questo con:

```
9C40               5         ORG  40000
9C40 2A409C       10         LD   HL,(CHANS)
     ;var.sistema
9C43 2B           20         DEC  HL
9C44 01449C       30         LD   BC,n
     ;n=num.bytes area necessaria
9C47 CD5516       40         CALL 5717
     ;SPACE-ACQ1
                  50 ;ora si puo' usare l'area
      da 23734 a 23734+n-1.
```

È anche possibile usare un'area allocata in zone alte di memoria, ma in tal caso, affinchè la routine funzioni correttamente, è necessario che i nomi delle variabili richiamate nello statement INPUT siano già esistenti, in modo da dovere es-

sere soltanto riassegnate; inoltre, in questo caso, se necessita poter rientrare al BASIC, la variabile CHADD deve essere reimpostata, in modo da puntare a una locazione contenente un codice NL dec. 13 (per es. WORKSP-2), oppure prima salvata e poi ripristinata.

Altre routines

Altre routines

INIT (dec 0 hex 00)

è la routine che viene automaticamente eseguita all'accensione del computer, ed esegue un completo IPL *(initial program loading)*.

Non ha senso un suo utilizzo all'interno di un programma, ma può risultare utile per evitare l'usura della presa di alimentazione dello Spectrum: anzichè disinserire e reinserire lo spinotto, quando ciò è necessario, basta un RANDOMIZE USR 0.

ERROR (RST dec 8 hex 08)

routine di *restart* in pagina 0, che causa la fine immediata del programma in esecuzione, evidenziandone la causa con un messaggio di errore *(report)*.

Viene chiamata con l'istruzione RST 8, seguita da un byte contenente il codice errore -1.

Per esempio.

RST 8: DEFB 10 interrompe l'esecuzione con messaggio "B integer out of range"

RST 8: DEFB 255, con "0 OK"

PAUSE (CALL dec 7997 hex 1F3D)

esegue l'analogo comando basic, ritornando solo dopo pressione di un tasto, o a tempo scaduto.

reg. in:

BC = valore del tempo di pausa (stesso operando del comando basic)

reg. out:

A e BC modificati

altri non modificati

TABLE-SEARCH (CALL dec 3137 hex 0C41)

ritrova un determinato elemento in una tabella di elementi

a lunghezza variabile.

reg. in:
A = numero dall'elemento cercato (0 per il primo)
DE = indirizzo inizio tabella
reg. out:
DE = indirizzo dell'elemento cercato
altri non modificati

Nota: il primo byte della tabella deve contenere un qualsiasi CODE uguale o maggiore di 128, che viene bypassato; l'ultimo byte di ciascun elemento, per essere riconosciuto (essendo variabile la lunghezza), deve avere il bit 7 ON (= CODE del carattere + 128).

CONV-DB (CALL dec 10929 e 11682 hex 2AB1 e 2DA2) esegue la conversione in binario di un numero decimale ASCII

La prima routine (STACK-STORE -5) memorizza il contenuto dei reg. DE e BC nel *calculator stack*; la seconda (STACK TO-BC) carica in BC il valore binario del LV, risultante dall'elaborazione della stringa da parte del calcolatore *floating pont.*

Occorre eseguire le seguenti istruzioni:

CALL 10929	STACK-STORE -5
LD B, 29	indica elaborazione di tipo stringa
RST 40:DEFBs 59,56	elabora la stringa
CALL 11682	STACK TO-BC

reg. in:
DE = indirizzo dell'area contenente la stringa decimale.
BC = numero di cifre = numero di bytes della stringa

reg. out:
BC = valore binario (max 65535)
IX non modificato
altri modificati
flags out:
C = valore originario maggiore di 65535

DATA-PASS (CALL dec 11249 hex 2BF1)

In pratica la routine (vedi STACK-FETCH) ha molti utilizzi, in quanto trasferisce semplicemente il LV del *calculator stack* nei registri A-E-D-C-B, ma in questo caso può servire ottimamente per passare dati da un programma basic a un programma in linguaggio macchina.

Esempio

Basic: LET x $ = "stringa": RANDOMIZE x $ = STR$ USR indcm

oppure RANDOMIZE "stringa" = STR$ USR indcm

C.M.: indcm CALL 11249

.... rientrando qui, DE e BC sono impostati

reg. out:

DE = indirizzo della stringa

BC = lunghezza della stringa

IX = non modificato

altri modificati

COMM-EX (CALL dec 7050 hex 1B8A)

esegue la linea basic precedentemente memorizzata nell'*edit-area* (ELINE) senza numero di linea; l'ultimo statement deve essere seguito dal codice NL dec. 13.

variab. in:

FLAGS (23611) con bit 7 ON indicante "run"

NSPPC (23620) = 1 per iniziare l'esecuzione dal primo statem. della linea, oppure *n* per iniziare dall' *n*-esimo statement

Esempio di uso:

```
9C40              5        ORG  40000
9C40 2A409C       10       LD   HL,(ELINE)
     ;var.sist.
9C43 E5           20       PUSH HL
9C44 016400       30       LD   BC,100
9C47 C5           40       PUSH BC
9C48 CD5516       50       CALL 5717
     ;SPACE-ACQ1: 100 bytes di edit-area
```

```
9C4B 214B9C        LD   HL,indir
     ;area con la linea basic
9C4E C1            POP  BC
9C4F D1            POP  DE
9C50 EDB0          LDIR ;trasf.linea
     basic in edit-area
9C52 FDCB01FE      SET  7,(IY+1)
     ;FLAGS
9C56 FD360A01      LD   (IY+10),1
     ;NSPPC
9C5A 2A5A9C        LD   HL,(NXTLIN)
     ;salva NXTLIN per tornare al basic
9C5D E5            PUSH HL
9C5E CD8A1B        CALL 7050
     ;comm-ex
9C61 D9            EXX  ;reimposta HL'
9C62 215827        LD   HL,10072
9C65 D9            EXX
9C66 E1            POP  HL
     ;ripristina HL
9C67 225A9C        LD   (NXTLIN),HL
9C6A C9            RET
```

Note: occorre acquisire lo spazio necessario nell'*edit-area*, e caricarvi la linea basic + NL.

Se necessita poter rientrare al BASIC, occorre salvare e ripristinare la variabile NXTLIN (23637) e reimpostare il registro HL' con l'indirizzo 10072.

La linea basic deve essere memorizzata nel formato consueto, con i valori numerici seguiti dalla corrispondente rappresentazione in virgola mobile; un metodo pratico per ricostruire correttamente una linea basic consiste nel digitarla normalmente, per esempio:

10 INPUT "raggio = "; R : CIRCLE 128, 88, R:
PRINT "superficie = "; R↑2 * PI

quindi aggiungere un programmino di lista decimale dei singoli bytes che la compongono:

20 FOR n = 23759 TO 24000 : PRINT PEEK n; " "; :
IF PEEK n = 13 THEN STOP
21 NEXT n

e infine dare RUN 20.

Normalmente, le rappresentazioni numeriche in virgola mobile vengono inserite in una linea basic dal *syntax checker* : è perciò possibile richiamarlo, per ottenere quel risultato.

La routine chiamante precedentemente esemplificata deve essere modificata, inserendo dopo il trasferimento della linea nell'*edit-area* (e prima di SET 7, (IY + 1)) una CALL 6935 : il *syntax checker* chiamato, oltre a controllare la sintassi, espande la linea basic, inserendovi la rappresentazione *floating point* di ogni numero incontrato.

Si può provare la seguente routine:

```
9C40               10         ORG  40000
9C40 FDCB37EE      20 start   SET  5,(IY+55)
     ;FLAGX
9C44 CDB016        30         CALL 5808
     ;CLEAR-EICS
9C47 010100        40         LD   BC,1
     ;WORKSP-acq
9C4A F7            50         RST  48
9C4B 360D          60         LD   (HL),13
9C4D 225B5C        70         LD   (23643),HL
     ;KCUR
9C50 AF            80         XOR  A
9C51 CD0116        90         CALL 5633
     ;apre il canale "k"
9C54 CD2C0F       100         CALL 3884
     ;INPUT
9C57 2A595C       110         LD   HL,(23641)
     ;ELINE
```

```
9C5A 016400      120        LD   BC,100
9C5D E5          130        PUSH HL
9C5E C5          140        PUSH BC
9C5F CD5516      150        CALL 5717
     ;SPACE-ACQ1 100bytes di edit-area
9C62 2A615C      160        LD   HL,(23649)
     ;WORKSP
9C65 C1          170        POP  BC
9C66 D1          180        POP  DE
9C67 EDB0        190        LDIR ;WORKSP to EL
     INE
9C69 CD171B      200        CALL 6935
     ;syntax-check
9C6C FDCB01FE    210        SET  7,(IY+1)
     ;FLAGS
9C70 FD360A01    220        LD   (IY+10),1
     ;NSPPC
9C74 CD8A1B      230        CALL 7050
     ;comm-ex
9C77 18C7        240        JR   start
```

assemblarla, eseguirla e digitare comandi a piacere: verranno eseguiti! Immettere il comando STOP per terminare.

Per poter digitare i comandi, anteporre THEN e poi cancellarlo.

La memorizzazione dei programmi BASIC e delle variabili

La memorizzazione dei programmi BASIC e delle variabili

Le due aree riservate ai programmi BASIC e alle variabili di programma non hanno in RAM una ubicazione prefissata, così come è variabile la loro estensione.

Tra l'area riservata alle variabili di sistema e l'area basic, vi possono essere una o due zone: *la microdrives maps a.* presente solo collegando i microdrives e relativa interfaccia, e la *channel informations a.*, sempre presente.

L'indirizzo di inizio dell'area basic è in ogni caso reperibile nella variabile PROG.

Le linee di programma vengono memorizzate nella loro esatta sequenza. Ciascuna linea, in memoria, è composta da quattro campi:

a = numero di linea, di 2 bytes (attenzione: alto + basso)
b = lunghezza in bytes di c + d, di 2 bytes
c = comandi, così come sono stati digitati, compresa la punteggiatura, con l'aggiunta, dopo ogni dato numerico, di ulteriori 6 bytes contenenti l'espressione floating point del numero (caratt. controllo 14 + 5 bytes f.p.). Ogni parola chiave basic o *token* (comando, specifico o funzione), viene memorizzata con un solo byte, pari al cod. ASCII corrispondente.
d = codice di ENTER o new line (13)

Per esempio la linea:
300 FOR N = 1 TO 5 : PRINT N; : NEXT N
risulta in memoria:

1 44	26 0	235	78	61	49	14	0 0 1 0 0
n. linea	lungh.	FOR	N	=	1	c.c.	valore 1 f.p.

204	53	14	0 0 5 0 0	58	245	78	59	58
TO	5	c.c.	valore 5 f.p.	:	PRINT	N	;	:

243	N	78 13
NEXT	N	ENTER

l'area in cui vengono registrate le variabili di programma segue immediatamente l'area basic; l'indirizzo di inizio è reperibile nella variabile VARS. L'area termina con un byte contenente il codice decimale 128, e l'indirizzo del byte successivo (inizio della *edit area)* è reperibile nella variabile ELINE.

Analizzando il contenuto dell'area variabili, occorre distinguere fra i diversi tipi: a stringa, numeriche, FOR-NEXT, arrays.

Si tenga presente che, ove si menziona *valore f.p.*, si intende un valore numerico rappresentato in una delle due possibili forme binarie floating point: full o semplificata, come descritto nel capitolo sul calcolatore f.p.

Variabili a stringa:
a b c1 ... cn
a = nome in codice ASCII maiuscolo
(1 byte: il simbolo $ non viene registrato)
b = lunghezza della stringa (2 b.: basso + alto)
c1 ... cn = stringa

es.: LET a$ = "a1B" è codificata con:

65	3	0	97	49	66
A			a	1	B

variabili numeriche con nome di 1 carattere:
a v

a = nome in codice ASCII minuscolo (1 byte)
v = valore f.p. (5 bytes)

es.:

LET A = 300	→	97	0	0	44	1	0
		"a"					
LET A = — 300	→	97	0	255	212	254	0
LET a = 123456	→	97	145	113	32	0	0

variabili numeriche con nome di più caratteri:
a1 ... an v

a1 = primo carattere del nome in codice ASCII minuscolo + 64
... = caratteri intermedi in codice ASCII (minuscolo se lettera)
an = ultimo carattere del nome in cod. ASCII (minuscolo se lettera) + 128
v = valore f.p. (5 bytes)
es.:

```
LET AB1C = 300    →   161 98 49 227 0 0 44 1 0
                      "a" + 64 "b" "1" "c" + 128
LET AbC1 = 300    →   161 98 99 177 0 0 44 1 0
                      "a" + 64 "b" "c" "1" + 128
```

Variabili FOR-NEXT:

a v1 v2 v3 b c

a = nome (1 byte) in codice ASCII minuscolo + 128
v1 = valore f.p. (5 b.) del corrente valore FOR
v2 = valore f.p. (5 b.) del valore TO
v3 = valore f.p. (5 b.) del valore STEP (default 1)
b = numero linea programma ove inizia il ciclo (2 bytes: basso + alto)
c = numero statement nella linea (1 byte)

```
es.: 100 FOR A=6 TO 300 STEP 2
     255       0 0 6 0 0  0 0 44 1 0  0 0 2 0 0   100 0    2

   "a"+128     FOR          TO          STEP      line n.  stat.
```

I bytes relativi a For contengono il valore 6 all'inizio, quindi assumono valore 8 al secondo passaggio, fino a 302 in uscita dal ciclo.

Arrays numerici:

a b c d1 ... dn v1 ... vn

a = nome (1 byte) in codice ASCII maiuscolo + 64
b = lunghezza in bytes di *c* + *d1* ÷ *dn* + *v1* ÷ *vn* (2 bytes: basso + alto)
c = numero dimensioni dell'array (1 byte)
d1 ... dn = numero elementi in ciascuna dimensione (per

ognuna dimensione, 2 bytes: basso + alto)
v1 ... vn = valori f.p. (5 b.) di ciascun elemento

es.: DIM a (2,5,4)

129	207 0	3	2 0	5 0	4 0
"A" + 64	lungh.	n.dim.	n.el	n.el	n.el

elem1 elem2 ... elem40
40*5 = 200 bytes

notare che tutti gli elementi risultano inizializzati con valore 0.

Arrays a stringa:
a b c d1 ... dn e1 ... en

a = nome (1 byte) in codice ASCII maiuscolo + 128
b = lunghezza in bytes di *c* + *d1* ÷ *dn* + *e1* ÷ *en* (2 bytes: basso + alto)
c = numero dimensioni dell'array (1 byte)
d1 ... dn = valore di ciascuna dimensione (per ognuna dimensione, 2 bytes: basso + alto)
e1 ... en = stringhe di ogni elemento, ciascuna della lunghezza fissa definita con l'ultima dimensione indicata in DIM: se una sola dimensione è stata indicata, viene assunta lunghezza 1 per default.

es.: DIM a$ (5,10)

193	55 0	2	5 0	10 0
"A" + 128	lungh.	n.dim.	dim.1	dim.2

elem1 elem2 ... elem5
5*10 = 50 bytes

DIM a$ (10)
193 13 0 1 10 0
elem1 elem2 ... elem10
10*1 = 10 bytes

Tutti gli elementi risultano inizializzati con *space* (cod. ASCII 32)

Una particolare caratteristica differenzia il trattamento delle variabili semplici a stringa: mentre le variabili di qualsiasi altro tipo mantengono costante, nell'area, la propria posizione relativa rispetto alle altre, nell'ordine naturale di creazione, le variabili semplici a stringa sono cancellate e riallocate sempre in coda ogni volta che vengono reimpostate da una specifica (LET o INPUT) anche quando la loro lunghezza non viene modificata: ciò comporta un corrispondente shift di compattamento delle altre variabili nell'area.

L'area di editing

Edit-area: l'area di edit e immissione

È questa l'area di lavoro utilizzata dal sistema per memorizzare, di volta in volta, i comandi da eseguire direttamente e le linee di programma basic, sia in fase di immissione, sia in fase di edit.

Da questa zona di *parcheggio*, le linee di programma vengono inserite ordinatamente nell'area basic, mentre i comandi sono immediatamente interpretati ed eseguiti. In entrambi i casi, i dati immessi vengono prima analizzati dal *syntax checker*, e ulteriormente elaborati solo se sintatticamente corretti.

Anche quest'area non ha una locazione prefissata in memoria: essendo ubicata immediatamente in coda ad altre aree dinamiche (*basic* e *variabili*), è destinata a seguirne le evoluzioni, slittando avanti o indietro secondo i casi.

L'indirizzo di inizio dell'area è costantemente aggiornato nella variabile di sistema ELINE (23641-23642).

L'area termina sempre con due bytes contenenti i valori decimali 13 (ENTER) e 128; il byte successivo è indirizzato dalla variabile WORKSP (23649-23650). Anche la sua estensione varia con molta dinamicità, in funzione della lunghezza dei dati contenuti.

Cercherò ora di spiegare come sia possibile utilizzare queste informazioni nei programmi in codice macchina, in modo pratico e originale.

Anche per un programmatore esperto, ottenere particolari funzioni in c.m. è un impegno non da poco; si pensi, per esempio, alla specifica basic CIRCLE o, più semplicemente, a INPUT: acquisire una stringa immessa da tastiera, un carattere dietro l'altro, certo non è un grosso problema; ma INPUT è molto più complesso, consentendo, prima dell'ENTER, ogni tipo di correzione di quanto già digitato.

È possibile risolvere il problema lasciando l'incombenza all'interprete basic, rimanendo tuttavia all'interno del programma in linguaggio macchina.

La soluzione consiste nell'acquisire un'area di immissione ampia a sufficienza per contenere il comando da eseguire, trasferirvi la stringa di codici ASCII che lo rappresentano (esattamente come fa il sistema con un comando immesso da tastiera), e consegnare il tutto al *command executor* affinchè lo esegua per nostro conto.

Naturalmente occorre qualche accorgimento; e siccome più della teoria e delle parole, vale la pratica, faccio seguire un'esauriente esemplificazione.

```
   1 REM "EDIT-AREA"
   2 REM  TEST & SAMPLE PROGRAM
  10 CLEAR 29999: FOR n=30000 TO 30198:
READ x: POKE n,x: NEXT n
  20 DATA 205,176,22,1,80,0,42,89,92,205
,85,22,42.85,92,229
  21 DATA 33,139,117,205,116,117,24,6,33
,160,117,205,116,117,33,153,117,205,116,
117,33,205,117,205,116,117
  22 DATA 42,89,92,43,43,43,126,246,32,2
54,115,40,225,254,110,32,233,33,88,39,21
7,225,34,85,92,201
  23 DATA 237,91,89,92,1,80,0,237,176,20
5,23,27,253,203,1,254,253,54,10,1,195,13
8,27
  24 DATA 229,51,48,48,58,227,120,58,227
,121,58,227,114,13
  25 DATA 216,120,44,121,44,114,13
  26 DATA 238,34,99,111,111,114,100,46,1
11,114,46,32,34,59,120,44,34,99,111,111,
114,100,46,118,101,114,116,46,32,34,59,1
21,44,34,114,97,103,103,105,111,32,34,59
,114,13
  27 DATA 238,34,118,117,111,105,32,99,1
11,110,116,105.110,117,97,114.101,63,32,
```

```
40,115,105,47,110,111,41,32,34,59,202,97
,36,58,245,172,48,44,48,59,97,36,13
 100 RANDOMIZE USR 30000
 110 PRINT AT 0,0;"*** fine ***": STOP
 300 REM coord.e raggio iniziale
 301 DATA 128,86,50
```

Il programma in c.m. dapprima legge i 3 parametri forniti con statement DATA alla linea 300 del basic (è utilizzabile quindi in generale per passaggio dati da basic a c.m.), e con questi valori disegna un cerchio; poi richiede l'immissione di ulteriori parametri per disegnare altri cerchi a piacere.

```
30000            20        ORG  30000
30000  CDB016    30        CALL 5808
30003  015000    40        LD   BC,80
30006  2A595C    50        LD   HL,(23641)
30009  CD5516    60        CALL 5717
30012  2A555C    70        LD   HL,(23637)
30015  E5        80        PUSH HL
30016  218B75    90        LD   HL,cinque
30019  CD7475   100        CALL quattr
30022  1806     110        JR   due
30024  21A075   120 uno    LD   HL,sette
30027  CD7475   130        CALL quattr
30030  219975   140 due    LD   HL,sei
30033  CD7475   150        CALL quattr
30036  21CC75   160 tre    LD   HL,otto
30039  CD7475   170        CALL quattr
30042  2A595C   180        LD   HL,(23641)
30045  2B       190        DEC  HL
30046  2B       200        DEC  HL
30047  2B       210        DEC  HL
30048  7E       220        LD   A,(HL)
30049  F620     230        OR   32
```

```
30051  FE73      240          CP   115
30053  28E1      250          JR   Z,uno
30055  FE6E      260          CP   110
30057  20E9      270          JR   NZ,tre
30059  215827    280          LD   HL,10072
30062  D9        290          EXX
30063  E1        300          POP  HL
30064  22555C    310          LD   (23637),HL
30067  C9        320          RET
30068  ED5B595C  330 quattr   LD   DE,(23641)
30072  015000    340          LD   BC,80
30075  EDB0      350          LDIR
30077  CD171B    360          CALL 6935
30080  FDCB01FE  370          SET  7,(IY+1)
30084  FD360A01  380          LD   (IY+10),1
30088  C38A1B    390          JP   7050
30091  E5        400 cinque   DEFB 229
30092  33        410          DEFB 51
30093  30        420          DEFB 48
30094  30        430          DEFB 48
30095  3A        440          DEFB ":"
30096  E3        450          DEFB 227
30097  78        460          DEFB "x"
30098  3A        470          DEFB ":"
30099  E3        480          DEFB 227
30100  79        490          DEFB "y"
30101  3A        500          DEFB ":"
30102  E3        510          DEFB 227
30103  72        520          DEFB "r"
30104  0D        530          DEFB 13
30105  D8        540 sei      DEFB 216
30106  782C792C  550          DEFM "x,y,r"
30111  0D        560          DEFB 13
30112  EE        570 sette    DEFB 238
30113  22        580          DEFB 34
```

```
30114 636F6F72    590          DEFM "coord.or."
30123 20          600          DEFB 32
30124 22          610          DEFB 34
30125 3B782C      620          DEFM ";x,"
30128 22          630          DEFB 34
30129 636F6F72    640          DEFM "coord.vert."

30140 22          650          DEFB 34
30141 3B792C      660          DEFM ";y,"
30144 22          670          DEFB 34
30145 72616767    680          DEFM "raggio "
30152 22          690          DEFB 34
30153 3B72        700          DEFM ";r"
30155 0D          710          DEFB 13
30156 EE          720 otto     DEFB 238
30157 22          730          DEFB 34
30158 76756F69    740          DEFM "vuoi continu
      are? (si/no) "
30183 22          750          DEFB 34
30184 3B          760          DEFB 59
30185 CA          770          DEFB 202
30186 61243A      780          DEFM "a$:"
30189 F5          790          DEFB 245
30190 AC          800          DEFB 172
30191 302C303B    810          DEFM "0,0;a$"
30197 0D          820          DEFB 13
```

Digitate il programma basic (listato 1), che carica il c.m. (disassemblato nel listato 2), e date RUN.

Una volta in memoria, il codice macchina può essere rieseguito semplicemente con RUN 100.

Qualche commento al programma assembly (listato 2):

30000 ÷ 30011: richiede il rilascio dell'area eventualmente esistente e acquisisce 80 bytes di *edit-area*. Nulla importa (a parte lo spreco di memoria) se l'area acquisita è superiore al necessario.

Si tenga pure presente che l'ampiezza dell'area rimarrà tale finchè non venga immesso da tastiera un comando o una linea di programma, o ne venga richiesto il rilascio.

30012 ÷ 30015: se si desidera poter rientrare al basic, la variabile NXTLIN deve essere salvata e, prima di RET, ripristinata.

30016 ÷ 30041: il registro HL viene caricato con l'indirizzo dell'area contenente lo statement che via via si vuole eseguire, e si invoca la routine per l'esecuzione

30068 ÷ 30090: Lo statement da eseguire viene trasferito in *edit-area*; anche in questo caso, non importa se si trasferiscono più bytes del dovuto, purchè non si superi la lunghezza dell'area acquisita: il primo codice di ENTER (dec 13) incontrato porrà termine ai comandi da eseguire presenti nell'area.

Quindi in successione: viene invocata la routine di *syntax check*, il bit 7 di 23611 (FLAGS) viene messo ON, a indicare esecuzione (OFF = controllo sintassi), in 23620 (NSPPC) viene impostato il valore 1, a indicare esecuzione dal primo statement della linea, e infine si chiama la routine di *command execution*.

30042 ÷ 30058: ritornando dal comando

INPUT *"vuoi continuare, etc".*

occorre poter indirizzare le variabile *a$* per testare la risposta; qui si sfrutta la circostanza che una variabile a stringa è sempre riallocata in coda alle altre variabili e nel caso specifico deve avere lunghezza 2.

In circostanze diverse si dovrà ricorrere ad altri metodi: per esempio, scansione di ricerca nell'area variabili, con istruzioni tipo CPIR o CPDR.

30059 ÷ 30067: prima di rientrare al basic, è necessaria reimpostare il reg. HL' con l'indirizzo di una istruzione di *end calc.* (dec 10072), e ripristinare la variabile NXTLIN.

30091 ÷ 30104: comando RESTORE e READ DATA

30105 ÷ 30111: comando CIRCLE

30112 ÷ 30156: comando INPUT nuovi parametri

30157 ÷ 30198: comando INPUT continuazione.

Le routines di stampa

L'utilizzo delle routines di stampa

Tutte le operazioni di stampa si ottengono utilizzando una routine generalizzata, residente in ROM all'indirizzo decimale 2548.

Detta routine viene normalmente chiamata tramite un'apposita istruzione di *restart* in pagina 0:RST 16; si ottiene la stampa, alla *current print position* del dispositivo di output precedentemente selezionato, di un carattere, il cui CODE è contenuto nel registro A.

Ad eccezione di A, nessun registro viene alterato: il loro contenuto è automaticamente salvato e ripristinato.

Il canale di output desiderato deve essere precedentemente impostato: esiste, a questo scopo un'altra breve routine, che imposta adeguatamente alcune variabili di sistema, in base al contenuto del registro A:

LD A, *n*:CALL 5633

dove *n* = 0 o 1 per can. "k" (schermo inferiore)
n = 2 per can. "s" (schermo superiore)
n = 3 per can. "p" (printer)

La *current print position* è data, a seconda del tipo di output interessato, dalle variabili di sistema:

PPOSN - PRCC per printer,
SPOSN - DFCC per screen superiore,
SPOSNL - ECHOE - DFCCL per screen inferiore.

Esse vengono opportunamente incrementate e aggiornate eseguendo la routine di stampa; anche a questo fine, esiste però una routine specifica, utile per impostare una determinata *current print position*:

LD B, *r* : LD C, *c* : CALL 3545

ove *r* = 24 — *num. riga* (cioè 24 per riga 0 ... 3 per riga 21)
c = 33 — *num. colonna* (cioè 33 per col. 0 ... 2 per col. 31)

il contenuto del registro B viene ignorato per outputs su printer.

Esistono altre routines di servizio, di uso molto frequente:

CALL 3435 esegue un completo clear-screen.

CALL 3438 esegue il clear-screen solo della parte inferiore dello schermo.

CALL 3582 esegue lo scroll-up dello schermo.

CALL 3807 esegue il clear del *printer buffer*

CALL 3789 trasferisce fisicamente su printer il contenuto del *printer buffer*, e ne effettua poi automaticamente il clear tramite la routine a 3807.

CALL 3756 esegue il *copy* su printer dello schermo video.

Come detto inizialmente, la routine di stampa è completamente generalizzata: oltre a gestire tutti i tipi output previsti, è in grado di gestire il set completo di codici ASCII: numeri e lettere, caratteri grafici (standard e *user-defined*), ***tokens***, e tutti i codici per il controllo della tabulazione (***newline,*** AT, TAB, etc.) e degli attributi (INK, PAPER, INVERSE, etc.).

I codici di controllo, ovviamente, non vengono inviati in stampa, ma elaborati, per impostare opportune variabili di sistema, che influenzeranno le modalità di stampa dei successivi caratteri stampabili.

A questo punto emerge spontaneo il problema: come inviare in stampa una serie continua di dati, cioè una *stringa*.

Letteralmente, ciò si realizza invocando ripetutamente la routine e fornendo a ogni ciclo, nel reg. A, il successivo byte della stringa.

Ancora una volta, abbiamo a disposizione una routine di servizio già pronta:

LD BC, *n* : LD DE, *i* : CALL 8252

dove

n = num. dei bytes componenti la stringa

i = indirizzo della stringa

Ecco un semplice esempio:
In BASIC

```
 100>PRINT AT 10,5;"Esempio: "'INK 2;"ROSS
O",PAPER 0;"NERO",INK 0;PAPER 7;"NORMALE",
INVERSE 1;"INVERSE";INVERSE 0
```

In assembler:

```
9C40                5            ORG   40000
9C40 3E02          10            LD    A,2
9C42 CD0116        20            CALL  5633
9C45 013300        30            LD    BC,51
9C48 114F9C        40            LD    DE,str
9C4B CD3C20        50            CALL  8252
9C4E C9            60            RET
9C4F 16            70 str        DEFB  22
9C50 0A            80            DEFB  10
9C51 05            90            DEFB  5
9C52 4573656D     100            DEFM  "Esempio: "
9C5B 0D           110            DEFB  13
9C5C 10           120            DEFB  16
9C5D 02           130            DEFB  2
9C5E 524F5353     140            DEFM  "ROSSO"
9C63 06           150            DEFB  6
9C64 11           160            DEFB  17
9C65 00           170            DEFB  0
9C66 4E45524F     180            DEFM  "NERO"
9C6A 06           190            DEFB  6
9C6B 10           200            DEFB  16
9C6C 00           210            DEFB  0
9C6D 11           220            DEFB  17
9C6E 07           230            DEFB  7
9C6F 4E4F524D     240            DEFM  "NORMALE"
9C76 06           250            DEFB  6
9C77 14           260            DEFB  20
```

```
9C78 01          270        DEFB 1
9C79 494E5645    280        DEFM "INVERSE"
9C80 14          290        DEFB 20
9C81 00          300        DEFB 0
9C82 0D          310        DEFB 13
```

che tradotto in codice macchina, si può caricare in memoria e provare, eseguendo:

```
  10 CLEAR 29999: FOR x=30000 TO
 30065: READ y: POKE x,y: NEXT x
  20 DATA 62,2,205,1,22,1,51,0,1
7,63,117,205,60,32,201,22,10,5,6
9,115,101,109,112,105,111,58,13
  30 DATA 16,2,82,79,83,83,79,6,
17,0,78,69,82,79,6,16,0,17,7,78,
79,82,77,65,76,69,6,20,1,73,78,8
6,69,82,83,69,20,0,13
 100 RANDOMIZE USR 30000: STOP
```

Non è indispensabile, anche se spesso molto comodo, che i codici di controllo siano inclusi nella stringa da stampare; lo stesso risultato si può ottenere impostando preventivamente le relative variabili di sistema.

PRINT AT 10,5	=	LD B,14:LD C,28:CALL 3545
INK 2	=	LD (IY+85), 2
BRIGHT 1	=	SET 6, (IY+85)
OVER 1	=	SET 0, (IY+87)
INVERSE 1	=	SET 2, (IY+87)

Annotiamo a margine che il reg. IY, nello Spectrum, è stato utilizzato come puntatore dell'area variabili di sistema, e contiene l'indirizzo 23610, corrispondente alla variabile ERRNR.

Concludo con qualche osservazione sull'uso della stampante.

Ogni riga, prima di essere emessa in output, viene costruita, carattere per carattere, colonna per colonna, nel *printer*

buffet, che si trova in RAM, dall'indirizzo 23296.

La sua capienza, 256 bytes, consente di memorizzare i 32 caratteri di 1 riga.

Affinchè l'emissione in stampa (scarico e clear del buffer) avvenga fisicamente, occorre che la riga sia completata; ciò si verifica in due circostanze:

1. chiamando la routine di stampa per l'emissione del 33.mo carattere
2. inviando alla routine di stampa un codice di controllo indicante un cambio di riga: normalmente il codice 13 *(newline)*.

Come già menzionato, possiamo forzare l'emissione con CALL 3789.

Infine, prestate attenzione alla locazione 23681, dichiarata *not used*: contiene invece 91 (byte alto dell'indirizzo del *printer buffer*); l'alterazione di questo valore comporta la perdita dei dati da stampare e, peggio ancora, la possibile cancellazione di aree di RAM.

Il calcolatore floating-point

---

approfondimento

Il calcolatore Floating-point

Questa essenziale componente della ROM dello Spectrum consiste di una complessa serie di routines, atte ad eseguire tanto le operazioni numeriche e logiche, quanto le elaborazioni su stringhe.

La sua conoscenza rende disponibile un potente strumento di programmazione in assembly, con notevoli risparmi di tempo e, cosa non indifferente, di occupazione di memoria.

Il *calcolatore floating point* (da qui in poi *cfp*), viene invocato con l'istruzione di restart RST 40, seguita dai bytes indicanti la sequenza di operazioni da svolgere: questi 82 codici operazione *(literals)* possono essere considerati come un insieme di *macro-istruzioni*, con le quali è possibile definire i vari passi di esecuzione di veri e propri sottoprogrammi, con la possibilità di compiere salti di sequenza *(jumps)*, condizionati o incondizionati.

Il flusso di esecuzione viene interrotto da un codice di operazione 56 *(end of calc.)*, che comporta il rientro alla routine chiamante.

Il *calculator stack* è l'area di lavoro principale, gestita con tecnica LIFO *(Last in first out)*, in cui il *cfp* registra via via i risultati parziali, eliminando i valori utilizzati e sostituendoli con i nuovi ottenuti, fino a ricavare il/i valori finali.

Ogni valore memorizzato consta di 5 bytes, e conseguentemente la *calculator stack area* si incrementa o decrementa di 5 bytes ogni qualvolta un valore viene registrato o eliminato. Le variabili STKBOT e STKEND, costantemente aggiornate, indicano gli indirizzi di *inizio* e di *fine + 1* dell'area occupata dal *calculator stack*.
È utile osservare che l'eliminazione di un valore consiste semplicemente nel *decremento* del puntatore (variabile STKEND) di 5 bytes: conseguentemente, i dati restano disponibili finchè lo spazio non venga riutilizzato.

Il valore più recente, presente al punto più alto dell'area,

viene identificato come il *last value:* di seguito indicherò questo valore con *LV* e quello che lo precede con LV-1.

Alcune operazioni fanno anche uso della *work space area* (WORKSP) e della *memory area* (MEMBOT).

I dati numerici trattati dal *cfp*, e memorizzati nello stack, sono sempre in formato *floating point* di 5 bytes, ma nello Spectrum si distinguono due possibili rappresentazioni:

— *full f.p.*, valida per qualsiasi valore, intero o decimale, compreso tra 0.29387359E-38 e 1.701E+38, in cui il primo dei 5 bytes rappresenta l'esponente e gli altri 4 bytes la mantissa;

— *semplificata*, usata per numeri interi compresi tra —65536 e +65535, in cui il primo e il quinto dei 5 bytes sono sempre 0, il secondo è 0 per valori positivi e 255 per valori negativi, il terzo e quarto sono i bytes di basso e alto ordine del numero reale, se questo è positivo, o del suo complemento a 2, se è negativo.

Esempi:

+ 10	=	0	0	10	0	0
— 10	=	0	255	246	255	0
0	=	0	0	0	0	0
— 65536	=	0	255	0	0	0

Vi è poi una terza forma *compressa*, utilizzata dal codice di operazione 52 *(stack-data)*, che consente di rappresentare un valore fp con un numero ridotto di bytes, variabile da 2 a 5.

Dal formato *compresso* si ricava quello in virgola mobile *full* o *semplificato*, nel modo che segue:

1. si divide il primo byte per 64, ottenendo *q* (quoziente) e *r* (resto)
2. se *r* è diverso da 0, allore l'esponente è dato da *r* + 80, e i bytes rimanenti vanno a formare la mantissa
3. se *r* = 0, l'esponente è dato dal byte di successivo (il secondo) + 80, e la mantissa dai bytes rimanenti.
4. *q*+1 dà il numero di bytes che seguono, a formare la

mantissa di 4 bytes: i bytes non specificati assumono valore 0.

Esempi:

0	compr.	=	0	176	0		
	full	=	0	0	0	0	0
10	compr	=	64	176	0	10	
	sempl	=	0	0	10	0	0

E così è usato in ROM, mentre sarebbe più conveniente:

	compr.	=	52	32			
	full	=	132	32	0	0	0
—23456	compr	=	127	183	64		
	full	=	143	183	64	0	0
67890	compr	=	64	65	4	153	
	full	=	145	4	153	0	0
3.456	compr	=	242	93	47	26	160
	full	=	130	93	47	26	160

Risulta evidente come il formato *compresso* sia conveniente quando la mantissa termina con almeno uno zero; il risparmio di memoria è comunque risibile: in pratica, nella ROM dello Spectrum, sono stati risparmiati solo una quarantina di bytes, considerato che la routine per la conversione da compresso ad espanso ne utilizza per sè una ventina.

I dati a stringa vengono trattati registrando nello *stack* (sempre con valori di 5 bytes) i parametri che definiscono la stringa.

Sia in immissione che in emissione, i registri BC e DE vengono impostati rispettivamente con la lunghezza della stringa e l'indirizzo di memoria ove essa si trova; nei 5 bytes del valore registrato nello *stack* vengono copiati i registri A-E-D-C-B.

Nelle prossime pagine evidenzierò con *$* e con *LV$* le operazioni e i valori che trattano stringhe.

Singole operazioni possono essere eseguite anche impostandone il "literal" nel reg. B ed eseguendo poi l'operazione

codice 59.

E questo è anche il metodo da usare per poter eseguire le operazioni di comparazione, corrispondenti ai "literals" 9 ÷ 14 e 17 ÷ 22.

Esempio:	LD B, 22	$ equal
	RST 40	
	DEFB 59	fp-calc 2
	DEFB 56	end-calc

I literals

Codici operazione (literals)

0 *(jump if true)* equivale a
JR NZ, *d*
ed esegue il salto di +/— *d* literals se LV (più precisamente il terzo byte di LV) contiene un valore *vero* (cioè non zero).
Il valore di *d* è dato dal *literal* successivo.
Es. RST 40:DEFBS..., 0, 3, a, b, c, d, ... , 56
Quando si giunge al codice, 0, si salta al cod. operaz. *c* se LV non è 0, altrimenti si prosegue col cod. operaz. *a*

1 *(exchange)* i due ultimi valori, LV e LV-1, vengono interscambiati:
LV = LV-1 e LV-1 = LV

2 *(delete)* l'ultimo valore viene logicamente eliminato dallo stack, decrementando di 5 il puntatore (variabile STKEND)

3 *(subtract)* LV-1 e LV vengono sottratti algebricamente, ottenendo un nuovo
LV = (LV-1) + (—LV)

4 *(multiply)* LV-1 e LV vengono moltiplicati, ottenendo un nuovo
LV = (LV—1) * LV

5 *(divide)* LV-1 e LV vengono divisi, ottenendo un nuovo
LV = (LV—1) / LV

6 *(to-power)* LV-1 viene elevato alla potenza di LV, ottenendo un nuovo
LV = (LV—1) ↑ LV

7 *(or)* esegue l'unione logica
(LV-1) OR LV
ottenendo un nuovo
LV = LV-1 se LV = 0 oppure
LV = 1 se LV diverso da 0

8 *(and)* esegue l'intersezione logica
(LV-1) AND LV
ottenendo un nuovo
LV = 0 se LV = 0 oppure
LV = LV-1 se LV diverso da 0

9 *(less or equal)* compara LV-1 e LV, ottenendo un nuovo
LV = 1 se la condizione è vera ((LV-1) minore o uguale a LV) o
LV = 0 se la condizione è falsa

10 *(greater or equal)* come il precedente, con il test
(LV-1) maggiore o uguale a LV

11 *(not equal)* come il precedente, con test
(LV-1) diverso da LV

12 *(greater)* come il precedente, con test
(LV-1) maggiore di LV

13 *(less)* come il precedente, con test
(LV-1) minore di LV

14 *(equal)* come il precedente, con test
(LV-1) = LV

15 *(add)* LV-1 e LV vengono sommati, ottenendo un nuovo
LV = (LV-1) + LV

16 *($ and n)* esegue l'intersezione logica
(LV$—1) AND LV
ottenendo un nuovo
LV$ = nullo, se LV = 0 oppure
LV$ = LV$—1 se LV diverso da 0

17 *($ less or equal)* compara LV$—1 e LV$, ottenendo un nuovo
LV = 1 se la condizione è vera
((LV$—1) minore o uguale a LV$) o
LV = 0 se la condizione è falsa

18 *($ greater or equal)* come il precedente, con test
(LV$—1) maggiore o uguale a LV$

19 *($ not equal)* come il precedente, con test
(LV$—1) diverso da LV$

20 *($ greater)* come il precedente, con test
(LV$—1) maggiore di LV$

21 *($ less)* come il precedente, con Test
(LV$—1) minore di LV$

22 *($ equal)* come il precedente, con test
(LV$—1) = LV$

23 *($ add)* esegue
LET S$ = X$ + Y$
dove X$ e Y$ sono rispettivamente LV$ e LV$—1.
La stringa risultante viene costruita in WORKSP, e il nuovo LV$ ne contiene i parametri

24 *(val$)* esegue
LET S$ = VAL$ "X$"
dove X$ è LV$.
La stringa risultante viene sviluppata in WORKSP e il nuovo LV$ ne contiene i parametri.

25 *($ usr)* esegue
LET I = USR "g"
gove g è LV$ e deve essere una lettera tra *a* e *u* (riferimento di un carattere grafico).
L'indirizzo del primo degli 8 bytes del carattere grafico viene impostato nel registro BC e memorizzato nel nuovo LV.

26 *(read in)* legge un byte dal canale corrispondente al numero impostato in LV.
La stringa di 1 byte, o nulla, contenente quanto letto, viene posta in WORKSP e il nuovo LV$ ne contiene i parametri.

27 *(negate)* esegue la negazione di LV, invertendone il segno.

28 *(code)* esegue
LET C = CODE C$

ove C$ è LV$, ottenendo un nuovo LV contenente il risultato.

29 *(val)* esegue
LET V = VAL S$ dove S$ è LV$, ottenendo un nuovo LV contenente il risultato.

30 *(len)* esegue
LET L = LEN S$
ove S$ è LV$, ottenendo un nuovo LV contenente il risultato.

31 *(sine)* esegue
LET S = SIN X
ove *x* è LV, ottenendo un nuovo LV contenente il risultato.

32 *(cosine)* esegue
LET C = COS X
ove *x* è LV, ottenendo un nuovo LV contenente il risultato.

33 *(tangent)* esegue
LET T = TAN X
ove *x* è LV, ottenendo un nuovo LV contenente il risultato.

34 *(arcsine)* esegue
LET A = ASN X, c.s.

35 *(arccosine)* esegue
LET A = ACS, c.s.

36 *(arctangent)* esegue
LET A = ATN X, c.s.

37 *(logarithm)* esegue
LET L = LNX, c.s.

38 *(exponent)* esegue
LET E = EXP X, c.s.

39 *(integer)* esegue
LET I = INT X, c.s.

40 *(square root)* esegue
LET S = SQ, c.s.

41 *(sign)* esegue
LET S = SGN X,
ove *x* è LV ottenendo un nuovo
LV = 1 se LV positivo
LV = 0 se LV zero
LV = —1 se LV negativo

42 *(absolute)* esegue
LET X = ABS X
ove *x* è LV, semplicemente eliminando il segno di LV

43 *(peek)* esegue
LET P = PEEK X
ove *x* è LV, ottenendo un nuovo LV contenente il risultato.

44 *(in)* esegue l'istruzione assembler IN A, (C), dopo aver trasferito in BC l'indirizzo contenuto in LV, e carica nel nuovo LV il byte letto.

45 *(usr)* esegue la funzione
USR I
ove I è LV, chiamando in esecuzione la routine in codice macchina a quell'indirizzo.
Il nuovo LV viene caricato con il contenuto del reg. BC, così come risulta al rientro dalla routine chiamata.
Nota: se necessita poter rientrare al basic, il reg. HL' deve essere reimpostato con l'indirizzo 10072.

46 *(str$)* esegue
LET S$ = STR$ X
ove *X* è LV.
La stringa risultante viene sviluppata in WORKSP, e il nuovo LV$ ne contiene i parametri.

47 *(chr$)* esegue
LET C$ = CHR$ X
ove *x* è LV.

La stringa di 1 byte risultante è posta in WORKSP e il nuovo LV$ ne contiene i parametri.

48 *(not)* ottiene un nuovo
LV = 1 se LV = 0, oppure
LV = 0 se LV = è diverso da 0.

49 *(duplicate)* carica lo stack con un nuovo valore, duplicando l'esistente LV.

50 *(module)* calcola
X (mod. y)
ove X e Y sono rispettivamente LV-1 e LV, ottenendo un nuovo
LV = INT (X/Y) e un nuovo
LV-1 = X — INT (X/Y).

51 *(jump)* equivale a
JR *d*
ed esegue il salto incondizionato di più o meno *d literals.*
Il valore di *d* è dato dal *literal* successivo (vedi esempio al codice operazione 0).

52 *(stack data)* carica lo stack con un nuovo valore, equivalente al numero floating point ottenuto elaborando i 2,3,4 o 5 *literals* successivi, che debbono contenere un numero f.p. in formato compresso (cfr. paragrafo *numeri f.p.)*

53 *(djnz)* equivale a
DJNZ *d.*
Il contatore decrementato è però la variabile BREG (23655), anzichè il reg. B. Il valore di *d* è dato dal literal successivo (vedi esempio al codice operazione 0).

Invocando il "calc. f.p." (RST 40), la variab. BREG viene inizializzata con il valore del reg. B.

54 *(less 0)* testa LV, ottenendo un nuovo
LV = 1 se LV minore di 0 oppure
LV = 0 se LV è maggiore o uguale a 0.

55 *(greater 0)* testa LV ottenendo un nuovo
LV = 1 se LV è maggiore di 0, oppure
LV = 0 se LV è minore o uguale a 0.

56 *(end of calc.)* termina l'esecuzione del *calcolatore f.p.*, eseguendo un RET.

57 *(get argument)* converte l'argomento X di SIN X o COS X in un valore che diviene il nuovo LV.

58 *(truncate)* esegue il troncamento di LV al suo intero sopprimendo le cifre decimali.
Differisce dalla funzione INT, per il fatto che non effettua arrotondamento all'intero inferiore: cioè —2.8 risulta —2, a differenza di INT —2,8 che risulta —3.

59 *(fp-calc. 2)* consente di richiamare il *calcolatore f.p.* come sottoprogramma di se stesso, per eseguire singole operazioni. È usato nello sviluppo delle espressioni.
Il codice operazione da eseguire deve trovarsi impostato nel reg. B.

60 *(e to fp)* calcola un nuovo LV, contenente il numero equivalente a xE + y, ove x è LV e y deve trovarsi impostato nel reg. A.
Esemplificando:
2 E +3 risulta 2000 (=2*10↑3)
e:
2 E —3 risulta .002 (=2/10↑3).

61 *(restack)* se LV contiene un numero in formato *semplificato*, lo trasforma nel formato *full f.p.*

134 - 136 - 140 *(mains series)* generano le serie dei polinomi di Chebyshev usate per calcolare SIN, ATN, LN, EXP e le altre funzioni matematiche che da questi dipendono.

160 ÷ 164 *(stack constans)* caricano lo stack con il valore corrispondente alla costante:

0 se cod. operaz. 160
1 se cod. operaz. 161
1/2 se cod. operaz. 162

PI/2 se cod. operaz. 163
10 se cod. operaz. 164

192 ÷ 197 *(store in memory)* copiano il LV dallo stack nell'area di memoria (generalmente MEMBOT (23698)), il cui inizio è indirizzato dalla variab. MEM (23656). Il codice operazione dà il *displacement* relativo all'inizio dell'area in cui trasferire i 5 bytes:

	mem 0	mem 1	mem 2	mem 3	mem 4	mem 5
cod. op	192	193	194	195	196	197

224 ÷ 229 *(get from memory)* caricano lo stack con il valore copiato dall'area di memoria: il processo è analogo a quello dell'operazione precedente, ma inverso.

Per completezza, è interessante annotare che le operazioni di *jump* (codice operazione 0 - 51 - 53) consentono di saltare ad operazioni appartenenti ad altro set di literals: in effetti, il *displacement* + /— d indica la distanza del salto, in termini di numero di bytes.

Esempio:

....
RST 40
DEFBs ...., 0, 10, a, b, 56 - Il *jump if true* (10) ha come destinazione il cod. op. *e*.
CALL xxx
RST 40
DEFBs c, d, e, ..., 56
...

Per concludere, due esemplificazioni
Entrambe le routines, con procedimenti diversi, raggiungono lo stesso risultato di ottenere come LV nel *calculator stack* un numero floating point equivalente all'espressione esponenziale
x E — y

La seconda è la routine esistente in ROM all'indirizzo 11599, usata per conseguire il codice operazione 60 *(e to fp)*: è (inspiegabilmente) molto più complessa, ma nondimento didatticamente utile.

Le due routines richiedono il valore *x* presente come LV nello *stack* e il valore +/— impostato nel registro A (es. + 4 = dec. 4 e — 4 = dec. 252); ricordo che 39 è il massimo esponente consentito.

Esempio 1

```
9C40              10          ORG   40000
9C40 07           20 EtoFP    RLCA
9C41 07           30          RLCA
9C42 3002         40          JR    NC,save
     ;flag NC se y positivo
9C44 2F           50          CPL
9C45 3C           60          INC   A
     ;-y = ABS y
9C46 F5           70 save     PUSH  AF
     ;ABS y
9C47 21925C       80          LD    HL,23698
     ;MEMBOT=mem0
9C4A CD0B35       90          CALL  13579
     ;crea in mem0 n.fp: 0 se y pos.(NC)

                 100 ;o 1 se y neg.(C)
9C4D F1          110          POP   AF
     ;ABS y
9C4E CD282D      120          CALL  11560
     ;stack reg.A; stack:LV-1=x, LV=y
9C51 EF          130          RST   40
     ;stack: LV-1=x, LV=y
9C52 E0          140          DEFB  224
     ;get mem0; stack: x,y,mem0
```

```
9C53 01          150         DEFB 1
     ;exchange; stack: x,mem0,y
9C54 A4          160         DEFB 164
     ;stack 10; stack: x,mem0,y,10
9C55 01          170         DEFB 1
     ;exchange; stack: x,mem0,10,y
9C56 06          180         DEFB 6
     ;to power; stack: x,mem0,10^y
9C57 01          190         DEFB 1
     ;exchange; stack: x,10^y,mem0
9C58 00          200         DEFB 0
9C59 04          210         DEFB 4
     ;jump+4(=div) se LV=1 (exp.neg.)
                 220 ;stack: x,10^y
9C5A 04          230         DEFB 4
     ;multiply; stack: x*10^y
9C5B 33          240         DEFB 51
9C5C 02          250         DEFB 2
     ;jump+2(end)
9C5D 05          260 div     DEFB 5
     ;divide; stack: x/10^y
9C5E 38          270 end     DEFB 56
     ;end calc
9C5F C9          280         RET
```

```
9C40              10         ORG  40000
9C40 07           20         RLCA
9C41 07           30         RLCA
9C42 3002         40         JR   NC,SAVE
9C44 2F           50         CPL
9C45 3C           60         INC  A
9C46 F5           70 SAVE    PUSH AF
9C47 21925C       80         LD   HL,23698
9C4A CD0B35       90         CALL 13579
```

```
9C4D EF          100          RST  40
     ;stack: LV=x
9C4E A4          110          DEFB 164
     ;stack 10; stack: x,10
9C4F 38          120          DEFB 56
     ;end calc
9C50 F1          130          POP  AF
     ;ABS y
```

Qui inizia un *loop*, shiftando nel carry un bit di *y* a ogni iterazione *n*, fino a ridurre $y = 0$;

n = 0 TO 5, non essendo y maggiore di 39.

Contemporaneamente si sviluppa il risultato:

x viene moltiplicato o diviso per 10 ↑ (2 ↑ *n*)

```
9C51 CB3F        140 LOOP     SRL  A
     ;shift left y con bit 0 in carry
9C53 300D        150          JR   NC,TEND
     ;flag NC se bit 0=0
9C55 F5          160          PUSH AF
     ;y/2^(n+1) e flag Z/NZ
9C56 EF          170          RST  40
     ;stack: LV-1=x',LV=10^(2^n)
9C57 C1          180          DEFB 193
     ;store mem1; mem1=10^(2^n)
9C58 E0          190          DEFB 224
     ;get mem0; stack: x',10^(2^n),mem0
9C59 00          200          DEFB 0
     ;jump+4(DIV) se LV=1 (exp.neg.)
9C5A 04          210          DEFB 4
     ;stack: x',10^(2^n)
9C5B 04          220          DEFB 4
     ;multiply; stack: x'*10^(2^n)=x'
9C5C 33          230          DEFB 51
     ;jump+2(=GET)
```

```
9C5D 02          240          DEFB 2
9C5E 05          250 DIV      DEFB 5
     ;divide; stack:x'/10^(2^n)=x'
9C5F E1          260 GET      DEFB 225
     ;get mem1; stack: x',10^(2^n)
9C60 38          270          DEFB 56
     ;end calc
9C61 F1          280          POP  AF
     ;y/2^(n+1) e flag Z/NZ
9C62 2808        290 TEND     JR   Z,END
     ;flag Z se y=0
9C64 F5          300          PUSH AF
     ;y/2^(n+1)
9C65 EF          310          RST  40
     ;stack: LV-1=x', LV=10^(2^n)
9C66 31          320          DEFB 49
     ;duplicate; stack: x',10^(2^n),10^(
     2^n)
9C67 04          330          DEFB 4
     ;multiply; stack: x',10^(2^(n+1))
9C68 38          340          DEFB 56
     ;end calc
9C69 F1          350          POP  AF
     ;y/2^(n+1)
9C6A 18E5        360          JR   LOOP
9C6C EF          370 END      RST  40
     ;stack: LV-1=x',LV=10^(2^(n+1))
9C6D 02          380          DEFB 2
     ;delete; stack:x'
9C6E 38          390          DEFB 56
     ;end calc
9C6F C9          400          RET
```

Entrambe le routines possono essere verificate con POKE 23728, y : RANDOMIZE X = USR «TESTR» e:

```
9C40 EF          20      RST  40
9C41 31          30      DEFB 49
9C42 38          40      DEFB 56
9C43 3AB05C      50      LD   A,(23728)
9C46 CD519C      60      CALL EtoFP
9C49 3E02        70      LD   A,2
9C4B CD0116      80      CALL 5633
9C4E C3E32D      90      JP   11747
```

Le istruzioni dello Z80

*** istruzioni assembler Z80 ***

```
0000 221 142 d     ADC  A,(IX+d)
0003 253 142 d     ADC  A,(IY+d)
0006 143           ADC  A,A
0007 136           ADC  A,B
0008 137           ADC  A,C
0009 138           ADC  A,D
0010 139           ADC  A,E
0011 140           ADC  A,H
0012 141           ADC  A,L
0013 142           ADC  A,(HL)
0014 206 n         ADC  A,n
0016 237 74        ADC  HL,BC
0018 237 90        ADC  HL,DE
0020 237 106       ADC  HL,HL
0022 237 122       ADC  HL,SP
0024 221 134 d     ADD  A,(IX+d)
0027 253 134 d     ADD  A,(IY+d)
0030 135           ADD  A,A
0031 128           ADD  A,B
0032 129           ADD  A,C
0033 130           ADD  A,D
0034 131           ADD  A,E
0035 132           ADD  A,H
0036 133           ADD  A,L
0037 134           ADD  A,(HL)
0038 198 n         ADD  A,n
0040 9             ADD  HL,BC
0041 25            ADD  HL,DE
0042 41            ADD  HL,HL
0043 57            ADD  HL,SP
0044 221 9         ADD  IX,BC
0046 221 25        ADD  IX,DE
0048 221 41        ADD  IX,IX
0050 221 57        ADD  IX,SP
0052 253 9         ADD  IY,BC
0054 253 25        ADD  IY,DE
0056 253 41        ADD  IY,IY
0058 253 57        ADD  IY,SP
0060 221 166 d     AND  (IX+d)
0063 253 166 d     AND  (IY+d)
0065 167           AND  A
0067 160           AND  B
0068 161           AND  C
0069 162           AND  D
0070 163           AND  E
0071 164           AND  H
0072 165           AND  L
0073 166           AND  (HL)
0074 230 n         AND  n
```

```
0076  221  203  d  70   BIT   0,(IX+d)
0080  253  203  d  70   BIT   0,(IY+d)
0084  203  71           BIT   0,A
0086  203  64           BIT   0,B
0088  203  65           BIT   0,C
0090  203  66           BIT   0,D
0092  203  67           BIT   0,E
0094  203  68           BIT   0,H
0096  203  69           BIT   0,L
0098  203  70           BIT   0,(HL)
0100  221  203  d  78   BIT   1,(IX+d)
0104  253  203  d  78   BIT   1,(IY+d)
0108  203  79           BIT   1,A
0110  203  72           BIT   1,B
0112  203  73           BIT   1,C
0114  203  74           BIT   1,D
0116  203  75           BIT   1,E
0118  203  76           BIT   1,H
0120  203  77           BIT   1,L
0122  203  78           BIT   1,(HL)
0124  221  203  d  86   BIT   2,(IX+d)
0128  253  203  d  86   BIT   2,(IY+d)
0132  203  87           BIT   2,A
0134  203  80           BIT   2,B
0136  203  81           BIT   2,C
0138  203  82           BIT   2,D
0140  203  83           BIT   2,E
0142  203  84           BIT   2,H
0144  203  85           BIT   2,L
0146  203  86           BIT   2,(HL)
0148  221  203  d  94   BIT   3,(IX+d)
0152  253  203  d  94   BIT   3,(IY+d)
0156  203  95           BIT   3,A
0158  203  88           BIT   3,B
0160  203  89           BIT   3,C
0162  203  90           BIT   3,D
0164  203  91           BIT   3,E
0166  203  92           BIT   3,H
0168  203  93           BIT   3,L
0170  203  94           BIT   3,(HL)
0172  221  203  d  102  BIT   4,(IX+d)
0176  253  203  d  102  BIT   4,(IY+d)
0180  203  103          BIT   4,A
0182  203  96           BIT   4,B
0184  203  97           BIT   4,C
0186  203  98           BIT   4,D
0188  203  99           BIT   4,E
0190  203  100          BIT   4,H
0192  203  101          BIT   4,L
0194  203  102          BIT   4,(HL)
0196  221  203  d  110  BIT   5,(IX+d)
```

```
0200  253  203  d  110  BIT   5,(IY+d)
0204  203  111          BIT   5,A
0206  203  104          BIT   5,B
0208  203  105          BIT   5,C
0210  203  106          BIT   5,D
0212  203  107          BIT   5,E
0214  203  108          BIT   5,H
0216  203  109          BIT   5,L
0218  203  110          BIT   5,(HL)
0220  221  203  d  118  BIT   6,(IX+d)
0224  253  203  d  118  BIT   6,(IY+d)
0228  203  119          BIT   6,A
0230  203  112          BIT   6,B
0232  203  113          BIT   6,C
0234  203  114          BIT   6,D
0236  203  115          BIT   6,E
0238  203  116          BIT   6,H
0240  203  117          BIT   6,L
0242  203  118          BIT   6,(HL)
0244  221  203  d  126  BIT   7,(IX+d)
0248  253  203  d  126  BIT   7,(IY+d)
0252  203  127          BIT   7,A
0254  203  120          BIT   7,B
0256  203  121          BIT   7,C
0258  203  122          BIT   7,D
0260  203  123          BIT   7,E
0262  203  124          BIT   7,H
0264  203  125          BIT   7,L
0266  203  126          BIT   7,(HL)
0268  205  a  a         CALL  aa
0271  220  a  a         CALL  C,aa
0274  212  a  a         CALL  NC,aa
0277  252  a  a         CALL  M,aa
0280  244  a  a         CALL  P,aa
0283  236  a  a         CALL  PE,aa
0286  228  a  a         CALL  PO,aa
0289  204  a  a         CALL  Z,aa
0292  196  a  a         CALL  NZ,aa
0295  63                CCF
0296  221  190  d       CP    (IX+d)
0299  253  190  d       CP    (IY+d)
0302  191               CP    A
0303  184               CP    B
0304  185               CP    C
0305  186               CP    D
0306  187               CP    E
0307  188               CP    H
0308  189               CP    L
0309  190               CP    (HL)
0310  254  n            CP    n
0312  237  169          CPD
```

```
0314 237 185          CPDR
0316 237 161          CPI
0318 237 177          CPIR
0320 47               CPL
0321 39               DAA
0322 221 43           DEC   IX
0324 221 53  d        DEC   (IX+d)
0327 253 43           DEC   IY
0329 253 53  d        DEC   (IY+d)
0332 61               DEC   A
0333 5                DEC   B
0334 11               DEC   BC
0335 13               DEC   C
0336 21               DEC   D
0337 27               DEC   DE
0338 29               DEC   E
0339 37               DEC   H
0340 45               DEC   L
0341 53               DEC   (HL)
0342 43               DEC   HL
0343 59               DEC   SP
0344 243              DI
0345 16  d            DJNZ  d
0347 251              EI
0348 227              EX    (SP),HL
0349 221 227          EX    (SP),IX
0351 253 227          EX    (SP),IY
0353 8                EX    AF,AF'
0354 235              EX    DE,HL
0355 217              EXX
0356 118              HALT
0357 237 70           IM    0
0359 237 86           IM    1
0361 237 94           IM    2
0363 237 120          IN    A,(C)
0365 219 n            IN    A,(n)
0367 237 64           IN    B,(C)
0369 237 72           IN    C,(C)
0371 237 80           IN    D,(C)
0373 237 88           IN    E,(C)
0375 237 96           IN    H,(C)
0377 237 104          IN    L,(C)
0379 221 35           INC   IX
0381 221 52  d        INC   (IX+d)
0384 253 35           INC   IY
0386 253 52  d        INC   (IY+d)
0389 60               INC   A
0390 4                INC   B
0391 3                INC   BC
0392 12               INC   C
0393 20               INC   D
```

```
0394  19                INC   DE
0395  28                INC   E
0396  36                INC   H
0397  44                INC   L
0398  52                INC   (HL)
0399  35                INC   HL
0400  51                INC   SP
0401  237  170          IND
0403  237  186          INDR
0405  237  162          INI
0407  237  178          INIR
0409  233               JP    (HL)
0410  221  233          JP    (IX)
0412  253  233          JP    (IY)
0414  195  a  a         JP    aa
0417  218  a  a         JP    C,aa
0420  210  a  a         JP    NC,aa
0423  250  a  a         JP    M,aa
0426  242  a  a         JP    P,aa
0429  234  a  a         JP    PE,aa
0432  226  a  a         JP    PO,aa
0435  202  a  a         JP    Z,aa
0438  194  a  a         JP    NZ,aa
0441  24  d             JR    d
0443  56  d             JR    C,d
0445  48  d             JR    NC,d
0447  40  d             JR    Z,d
0449  32  d             JR    NZ,d
0451  237  71           LD    I,A
0453  221  42  n  n     LD    IX,(nn)
0457  221  33  n  n     LD    IX,nn
0461  221  119  d       LD    (IX+d),A
0464  221  112  d       LD    (IX+d),B
0467  221  113  d       LD    (IX+d),C
0470  221  114  d       LD    (IX+d),D
0473  221  115  d       LD    (IX+d),E
0476  221  116  d       LD    (IX+d),H
0479  221  117  d       LD    (IX+d),L
0482  221  54  d  n     LD    (IX+d),n
0486  253  42  n  n     LD    IY,(nn)
0490  253  33  n  n     LD    IY,nn
0494  253  119  d       LD    (IY+d),A
0497  253  112  d       LD    (IY+d),B
0500  253  113  d       LD    (IY+d),C
0503  253  114  d       LD    (IY+d),D
0506  253  115  d       LD    (IY+d),E
0509  253  116  d       LD    (IY+d),H
0512  253  117  d       LD    (IY+d),L
0515  253  54  d  n     LD    (IY+d),n
0519  50  n  n          LD    (nn),A
0522  237  67  n  n     LD    (nn),BC
```

```
0526 237 83 n n    LD   (nn),DE
0530 34 n n        LD   (nn),HL
0533 221 34 n n    LD   (nn),IX
0537 253 34 n n    LD   (nn),IY
0541 237 115 n n   LD   (nn),SP
0545 237 87        LD   A,I
0547 221 126 d     LD   A,(IX+d)
0550 253 126 d     LD   A,(IY+d)
0553 237 95        LD   A,R
0555 127           LD   A,A
0556 120           LD   A,B
0557 121           LD   A,C
0558 10            LD   A,(BC)
0559 122           LD   A,D
0560 123           LD   A,E
0561 26            LD   A,(DE)
0562 124           LD   A,H
0563 125           LD   A,L
0564 126           LD   A,(HL)
0565 58 n n        LD   A,(nn)
0568 62 n          LD   A,n
0570 221 70 d      LD   B,(IX+d)
0573 253 70 d      LD   B,(IY+d)
0576 71            LD   B,A
0577 64            LD   B,B
0578 65            LD   B,C
0579 66            LD   B,D
0580 67            LD   B,E
0581 68            LD   B,H
0582 69            LD   B,L
0583 70            LD   B,(HL)
0584 6 n           LD   B,n
0586 2             LD   (BC),A
0587 237 75 n n    LD   BC,(nn)
0591 1 n n         LD   BC,nn
0594 221 78 d      LD   C,(IX+d)
0597 253 78 d      LD   C,(IY+d)
0600 79            LD   C,A
0601 72            LD   C,B
0602 73            LD   C,C
0603 74            LD   C,D
0604 75            LD   C,E
0605 76            LD   C,H
0606 77            LD   C,L
0607 78            LD   C,(HL)
0608 14 n          LD   C,n
0610 221 86 d      LD   D,(IX+d)
0613 253 86 d      LD   D,(IY+d)
0616 87            LD   D,A
0617 80            LD   D,B
0618 81            LD   D,C
```

```
0619  82                 LD    D,D
0620  83                 LD    D,E
0621  84                 LD    D,H
0622  85                 LD    D,L
0623  86                 LD    D,(HL)
0624  22 n               LD    D,n
0626  18                 LD    (DE),A
0627  237 91 n n         LD    DE,(nn)
0631  17 n n             LD    DE,nn
0634  221 94 d           LD    E,(IX+d)
0637  253 94 d           LD    E,(IY+d)
0640  95                 LD    E,A
0641  88                 LD    E,B
0642  89                 LD    E,C
0643  90                 LD    E,D
0644  91                 LD    E,E
0645  92                 LD    E,H
0646  93                 LD    E,L
0647  94                 LD    E,(HL)
0648  30 n               LD    E,n
0650  221 102 d          LD    H,(IX+d)
0653  253 102 d          LD    H,(IY+d)
0656  103                LD    H,A
0657  96                 LD    H,B
0658  97                 LD    H,C
0659  98                 LD    H,D
0660  99                 LD    H,E
0661  100                LD    H,H
0662  101                LD    H,L
0663  102                LD    H,(HL)
0664  38 n               LD    H,n
0666  119                LD    (HL),A
0667  112                LD    (HL),B
0668  113                LD    (HL),C
0669  114                LD    (HL),D
0670  115                LD    (HL),E
0671  116                LD    (HL),H
0672  117                LD    (HL),L
0673  54 n               LD    (HL),n
0675  42 n n             LD    HL,(nn)
0678  33 n n             LD    HL,nn
0681  221 110 d          LD    L,(IX+d)
0684  253 110 d          LD    L,(IY+d)
0687  111                LD    L,A
0688  104                LD    L,B
0689  105                LD    L,C
0690  106                LD    L,D
0691  107                LD    L,E
0692  108                LD    L,H
0693  109                LD    L,L
0694  110                LD    L,(HL)
```

```
0695 46  n            LD    L,n
0697 237 79           LD    R,A
0699 237 123 n  n     LD    SP,(nn)
0703 249              LD    SP,HL
0704 221 249          LD    SP,IX
0706 253 249          LD    SP,IY
0708 49  n  n         LD    SP,nn
0711 237 168          LDD
0713 237 184          LDDR
0715 237 160          LDI
0717 237 176          LDIR
0719 237 68           NEG
0721 0                NOP
0722 221 182 d        OR    (IX+d)
0725 253 182 d        OR    (IY+d)
0728 183              OR    A
0729 176              OR    B
0730 177              OR    C
0731 178              OR    D
0732 179              OR    E
0733 180              OR    H
0734 181              OR    L
0735 182              OR    (HL)
0736 246 n            OR    n
0738 237 187          OTDR
0740 237 179          OTIR
0742 237 121          OUT   (C),A
0744 237 65           OUT   (C),B
0746 237 73           OUT   (C),C
0748 237 81           OUT   (C),D
0750 237 89           OUT   (C),E
0752 237 97           OUT   (C),H
0754 237 105          OUT   (C),L
0756 211 n            OUT   (n),A
0758 237 171          OUTD
0760 237 163          OUTI
0762 241              POP   AF
0763 193              POP   BC
0764 209              POP   DE
0765 225              POP   HL
0766 221 225          POP   IX
0768 253 225          POP   IY
0770 245              PUSH  AF
0771 197              PUSH  BC
0772 213              PUSH  DE
0773 229              PUSH  HL
0774 221 229          PUSH  IX
0776 253 229          PUSH  IY
0778 221 203 d  134   RES   0,(IX+d)
0782 253 203 d  134   RES   0,(IY+d)
0786 203 135          RES   0,A
```

```
0788 203 128           RES  0,B
0790 203 129           RES  0,C
0792 203 130           RES  0,D
0794 203 131           RES  0,E
0796 203 132           RES  0,H
0798 203 133           RES  0,L
0800 203 134           RES  0,(HL)
0802 221 203  d  142   RES  1,(IX+d)
0806 253 203  d  142   RES  1,(IY+d)
0810 203 143           RES  1,A
0812 203 136           RES  1,B
0814 203 137           RES  1,C
0816 203 138           RES  1,D
0818 203 139           RES  1,E
0820 203 140           RES  1,H
0822 203 141           RES  1,L
0824 203 142           RES  1,(HL)
0826 221 203  d  150   RES  2,(IX+d)
0830 253 203  d  150   RES  2,(IY+d)
0834 203 151           RES  2,A
0836 203 144           RES  2,B
0838 203 145           RES  2,C
0840 203 146           RES  2,D
0842 203 147           RES  2,E
0844 203 148           RES  2,H
0846 203 149           RES  2,L
0848 203 150           RES  2,(HL)
0850 221 203  d  158   RES  3,(IX+d)
0854 253 203  d  158   RES  3,(IY+d)
0858 203 159           RES  3,A
0860 203 152           RES  3,B
0862 203 153           RES  3,C
0864 203 154           RES  3,D
0866 203 155           RES  3,E
0868 203 156           RES  3,H
0870 203 157           RES  3,L
0872 203 158           RES  3,(HL)
0874 221 203  d  166   RES  4,(IX+d)
0878 253 203  d  166   RES  4,(IY+d)
0882 203 167           RES  4,A
0884 203 160           RES  4,B
0886 203 161           RES  4,C
0888 203 162           RES  4,D
0890 203 163           RES  4,E
0892 203 164           RES  4,H
0894 203 165           RES  4,L
0896 203 166           RES  4,(HL)
0898 221 203  d  174   RES  5,(IX+d)
0902 253 203  d  174   RES  5,(IY+d)
0906 203 175           RES  5,A
0908 203 168           RES  5,B
```

```
0910 203 169          RES   5,C
0912 203 170          RES   5,D
0914 203 171          RES   5,E
0916 203 172          RES   5,H
0918 203 173          RES   5,L
0920 203 174          RES   5,(HL)
0922 221 203  d  182  RES   6,(IX+d)
0926 253 203  d  182  RES   6,(IY+d)
0930 203 183          RES   6,A
0932 203 176          RES   6,B
0934 203 177          RES   6,C
0936 203 178          RES   6,D
0938 203 179          RES   6,E
0940 203 180          RES   6,H
0942 203 181          RES   6,L
0944 203 182          RES   6,(HL)
0946 221 203  d  190  RES   7,(IX+d)
0950 253 203  d  190  RES   7,(IY+d)
0954 203 191          RES   7,A
0956 203 184          RES   7,B
0958 203 185          RES   7,C
0960 203 186          RES   7,D
0962 203 187          RES   7,E
0964 203 188          RES   7,H
0966 203 189          RES   7,L
0968 203 190          RES   7,(HL)
0970 201              RET
0971 216              RET   C
0972 208              RET   NC
0973 248              RET   M
0974 240              RET   P
0975 232              RET   PE
0976 224              RET   PO
0977 200              RET   Z
0978 192              RET   NZ
0979 237 77           RETI
0981 237 69           RETN
0983 221 203  d  22   RL    (IX+d)
0987 253 203  d  22   RL    (IY+d)
0991 23               RLA
0992 203 23           RL    A
0994 203 16           RL    B
0996 203 17           RL    C
0998 203 18           RL    D
1000 203 19           RL    E
1002 203 20           RL    H
1004 203 21           RL    L
1006 203 22           RL    (HL)
1008 221 203  d  6    RLC   (IX+d)
1012 253 203  d  6    RLC   (IY+d)
1016 7                RLCA
```

```
1017 203 7            RLC  A
1019 203 0            RLC  B
1021 203 1            RLC  C
1023 203 2            RLC  D
1025 203 3            RLC  E
1027 203 4            RLC  H
1029 203 5            RLC  L
1031 203 6            RLC  (HL)
1033 237 111          RLD
1035 221 203 d 30     RR   (IX+d)
1039 253 203 d 30     RR   (IY+d)
1043 31               RRA
1044 203 31           RR   A
1046 203 24           RR   B
1048 203 25           RR   C
1050 203 26           RR   D
1052 203 27           RR   E
1054 203 28           RR   H
1056 203 29           RR   L
1058 203 30           RR   (HL)
1060 221 203 d 14     RRC  (IX+d)
1064 253 203 d 14     RRC  (IY+d)
1068 15               RRCA
1069 203 15           RRC  A
1071 203 8            RRC  B
1073 203 9            RRC  C
1075 203 10           RRC  D
1077 203 11           RRC  E
1079 203 12           RRC  H
1081 203 13           RRC  L
1083 203 14           RRC  (HL)
1085 237 103          RRD
1087 199              RST  0
1088 207              RST  8
1089 215              RST  16
1090 223              RST  24
1091 231              RST  32
1092 239              RST  40
1093 247              RST  48
1094 255              RST  56
1095 221 158 d        SBC  A,(IX+d)
1098 253 158 d        SBC  A,(IY+d)
1101 159              SBC  A,A
1102 152              SBC  A,B
1103 153              SBC  A,C
1104 154              SBC  A,D
1105 155              SBC  A,E
1106 156              SBC  A,H
1107 157              SBC  A,L
1108 158              SBC  A,(HL)
1109 222 n            SBC  A,n
```

```
1111  237  66                  SBC   HL,BC
1113  237  82                  SBC   HL,DE
1115  237  98                  SBC   HL,HL
1117  237  114                 SBC   HL,SP
1119  55                       SCF
1120  221  203  d   198        SET   0,(IX+d)
1124  253  203  d   198        SET   0,(IY+d)
1128  203  199                 SET   0,A
1130  203  192                 SET   0,B
1132  203  193                 SET   0,C
1134  203  194                 SET   0,D
1136  203  195                 SET   0,E
1138  203  196                 SET   0,H
1140  203  197                 SET   0,L
1142  203  198                 SET   0,(HL)
1144  221  203  d   206        SET   1,(IX+d)
1148  253  203  d   206        SET   1,(IY+d)
1152  203  207                 SET   1,A
1154  203  200                 SET   1,B
1156  203  201                 SET   1,C
1158  203  202                 SET   1,D
1160  203  203                 SET   1,E
1162  203  204                 SET   1,H
1164  203  205                 SET   1,L
1166  203  206                 SET   1,(HL)
1168  221  203  d   214        SET   2,(IX+d)
1172  253  203  d   214        SET   2,(IY+d)
1176  203  215                 SET   2,A
1178  203  208                 SET   2,B
1180  203  209                 SET   2,C
1182  203  210                 SET   2,D
1184  203  211                 SET   2,E
1186  203  212                 SET   2,H
1188  203  213                 SET   2,L
1190  203  214                 SET   2,(HL)
1192  221  203  d   222        SET   3,(IX+d)
1196  253  203  d   222        SET   3,(IY+d)
1200  203  223                 SET   3,A
1202  203  216                 SET   3,B
1204  203  217                 SET   3,C
1206  203  218                 SET   3,D
1208  203  219                 SET   3,E
1210  203  220                 SET   3,H
1212  203  221                 SET   3,L
1214  203  222                 SET   3,(HL)
1216  221  203  d   230        SET   4,(IX+d)
1220  253  203  d   230        SET   4,(IY+d)
1224  203  231                 SET   4,A
1226  203  224                 SET   4,B
1228  203  225                 SET   4,C
1230  203  226                 SET   4,D
```

```
1232  203  227             SET   4,E
1234  203  228             SET   4,H
1236  203  229             SET   4,L
1238  203  230             SET   4,(HL)
1240  221  203  d   238    SET   5,(IX+d)
1244  253  203  d   238    SET   5,(IY+d)
1248  203  239             SET   5,A
1250  203  232             SET   5,B
1252  203  233             SET   5,C
1254  203  234             SET   5,D
1256  203  235             SET   5,E
1258  203  236             SET   5,H
1260  203  237             SET   5,L
1262  203  238             SET   5,(HL)
1264  221  203  d   246    SET   6,(IX+d)
1268  253  203  d   246    SET   6,(IY+d)
1272  203  247             SET   6,A
1274  203  240             SET   6,B
1276  203  241             SET   6,C
1278  203  242             SET   6,D
1280  203  243             SET   6,E
1282  203  244             SET   6,H
1284  203  245             SET   6,L
1286  203  246             SET   6,(HL)
1288  221  203  d   254    SET   7,(IX+d)
1292  253  203  d   254    SET   7,(IY+d)
1296  203  255             SET   7,A
1298  203  248             SET   7,B
1300  203  249             SET   7,C
1302  203  250             SET   7,D
1304  203  251             SET   7,E
1306  203  252             SET   7,H
1308  203  253             SET   7,L
1310  203  254             SET   7,(HL)
1312  221  203  d   38     SLA   (IX+d)
1316  253  203  d   38     SLA   (IY+d)
1320  203  39              SLA   A
1322  203  32              SLA   B
1324  203  33              SLA   C
1326  203  34              SLA   D
1328  203  35              SLA   E
1330  203  36              SLA   H
1332  203  37              SLA   L
1334  203  38              SLA   (HL)
1336  221  203  d   46     SRA   (IX+d)
1340  253  203  d   46     SRA   (IY+d)
1344  203  47              SRA   A
1346  203  40              SRA   B
1348  203  41              SRA   C
1350  203  42              SRA   D
1352  203  43              SRA   E
```

1354	203	44			SRA	H
1356	203	45			SRA	L
1358	203	46			SRA	(HL)
1360	221	203	d	62	SRL	(IX+d)
1364	253	203	d	62	SRL	(IY+d)
1368	203	63			SRL	A
1370	203	56			SRL	B
1372	203	57			SRL	C
1374	203	58			SRL	D
1376	203	59			SRL	E
1378	203	60			SRL	H
1380	203	61			SRL	L
1382	203	62			SRL	(HL)
1384	221	150	d		SUB	(IX+d)
1387	253	150	d		SUB	(IY+d)
1390	151				SUB	A
1391	144				SUB	B
1392	145				SUB	C
1393	146				SUB	D
1394	147				SUB	E
1395	148				SUB	H
1396	149				SUB	L
1397	150				SUB	(HL)
1398	214	n			SUB	n
1400	221	174	d		XOR	(IX+d)
1403	253	174	d		XOR	(IY+d)
1405	175				XOR	A
1407	168				XOR	B
1408	169				XOR	C
1409	170				XOR	D
1410	171				XOR	E
1411	172				XOR	H
1412	173				XOR	L
1413	174				XOR	(HL)
1414	238	n			XOR	n

Conversioni decimali/ esadecimali

** tavola conversione dec/hex **

0	00	64	40	128	80	192	C0
1	01	65	41	129	81	193	C1
2	02	66	42	130	82	194	C2
3	03	67	43	131	83	195	C3
4	04	68	44	132	84	196	C4
5	05	69	45	133	85	197	C5
6	06	70	46	134	86	198	C6
7	07	71	47	135	87	199	C7
8	08	72	48	136	88	200	C8
9	09	73	49	137	89	201	C9
10	0A	74	4A	138	8A	202	CA
11	0B	75	4B	139	8B	203	CB
12	0C	76	4C	140	8C	204	CC
13	0D	77	4D	141	8D	205	CD
14	0E	78	4E	142	8E	206	CE
15	0F	79	4F	143	8F	207	CF
16	10	80	50	144	90	208	D0
17	11	81	51	145	91	209	D1
18	12	82	52	146	92	210	D2
19	13	83	53	147	93	211	D3
20	14	84	54	148	94	212	D4
21	15	85	55	149	95	213	D5
22	16	86	56	150	96	214	D6
23	17	87	57	151	97	215	D7
24	18	88	58	152	98	216	D8
25	19	89	59	153	99	217	D9
26	1A	90	5A	154	9A	218	DA
27	1B	91	5B	155	9B	219	DB
28	1C	92	5C	156	9C	220	DC
29	1D	93	5D	157	9D	221	DD
30	1E	94	5E	158	9E	222	DE
31	1F	95	5F	159	9F	223	DF
32	20	96	60	160	A0	224	E0
33	21	97	61	161	A1	225	E1
34	22	98	62	162	A2	226	E2
35	23	99	63	163	A3	227	E3
36	24	100	64	164	A4	228	E4
37	25	101	65	165	A5	229	E5
38	26	102	66	166	A6	230	E6
39	27	103	67	167	A7	231	E7
40	28	104	68	168	A8	232	E8
41	29	105	69	169	A9	233	E9
42	2A	106	6A	170	AA	234	EA
43	2B	107	6B	171	AB	235	EB
44	2C	108	6C	172	AC	236	EC
45	2D	109	6D	173	AD	237	ED
46	2E	110	6E	174	AE	238	EE
47	2F	111	6F	175	AF	239	EF
48	30	112	70	176	B0	240	F0

49	31	113	71	177	B1	241	F1
50	32	114	72	178	B2	242	F2
51	33	115	73	179	B3	243	F3
52	34	116	74	180	B4	244	F4
53	35	117	75	181	B5	245	F5
54	36	118	76	182	B6	246	F6
55	37	119	77	183	B7	247	F7
56	38	120	78	184	B8	248	F8
57	39	121	79	185	B9	249	F9
58	3A	122	7A	186	BA	250	FA
59	3B	123	7B	187	BB	251	FB
60	3C	124	7C	188	BC	252	FC
61	3D	125	7D	189	BD	253	FD
62	3E	126	7E	190	BE	254	FE
63	3F	127	7F	191	BF	255	FF

Tabella di conversione decimale/esadecimale

a) per convertire da esadecimale a decimale, trovate il numero decimale corrispondente a ciascuna cifra esadecimale nella rispettiva colonna (da I a VI), quindi sommate tra loro questi valori. Es.: convertire 9CC2E. I decimali corrispondenti sono 589824,49152, 3072,32,14; la loro somma da' 642094, che e' il numero cercato.

b) per convertire da decimale a esadecimale, trovate nella tabella il numero piu' grande inferiore a quello da convertire, e segnatevi la cifra esadecimale corrispondente; calcolate la differenza tra il vostro numero dec e quello scelto nella tabella, quindi cercate il piu' grande minore di tale resto, etc. Es.: convertire 57248. Il primo inferiore nella tabella e' 53248 (hex D); il resto e' 4000; i numeri successivi sono 3840 (F); 160 (A); 0 (0); il numero in formato hex e' percio' DFA0.

Conversione Decimale/Esadecimale

VI		V		IV		III		II		I	
HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC	H	D	H	D
0	0	0	0	0	0	0	0	0	0	0	0
1	1048576	1	65536	1	4096	1	256	1	16	1	1
2	2097152	2	131072	2	8192	2	512	2	32	2	2
3	3145728	3	196608	3	12288	3	768	3	48	3	3
4	4194304	4	262144	4	16384	4	1024	4	64	4	4
5	5242880	5	327680	5	20480	5	1280	5	80	5	5
6	6291456	6	393216	6	24576	6	1536	6	96	6	6
7	7340032	7	458752	7	28672	7	1792	7	112	7	7
8	8388608	8	524288	8	32768	8	2048	8	128	8	8
9	9437184	9	589824	9	36864	9	2304	9	144	9	9
A	10485760	A	655360	A	40960	A	2560	A	160	A	10
B	11534336	B	720897	B	45056	B	2816	B	176	B	11
C	12582912	C	786432	C	49152	C	3072	C	192	C	12
D	13631488	D	851968	D	53248	D	3328	D	208	D	13
E	14680064	E	917504	E	57344	E	3584	E	224	E	14
F	15728640	F	983040	F	61440	F	3840	F	240	F	15

Mappa della ROM

Mappa della ROM dello Spectrum

Routines di restart e tabelle

0000 start
0008 restart di errore
0010 stampa un carattere
0018 preleva un carattere
0020 preleva il prossimo carattere
0028 restart del calcolatore
0030 crea locazioni libere nell'area di lavoro
0038 interrupt mascherabile
0053 routine errore-2
0066 interrupt non mascherabile
0074 subroutine CH-ADD + 1
007D SKIP-OVER
0095 tabella dei token
0205 tabella dei tasti

Routines della tastiera

028E scansione della tastiera
02BF scans.tastiera e decodifica del valore
0310 ripetizione del tasto
031E controllo del tasto premuto
0333 decodifica della tastiera

Routines dell'altoparlante

03B5 controllo altoparlante
03F8 gestione del BEEP
046E tabella dei semitoni

Routines di gestione del registratore

04C2 salva bytes di header e blocco program/data
053F reset cornice, controllo tasto BREAK durante LOAD/SAVE
0556 carica header e blocco informazioni
05E3 controlla il segnale in caricamento
0605 entry-point per comandi del nastro
07CB routine di controllo di VERIFY

0802 carica un blocco di dati
0808 rout.di controllo di LOAD
08B6 rout.di controllo di MERGE
0970 rout.di controllo di SAVE
09A1 messaggi fissi sulla cassetta

Routines di gestione schermo e stampante

09F4 routine generale di PRINT
0A11 tabella dei caratteri di controllo
0A23 cursore a sinistra
0A3D cursore a destra
0A4F "carriage return"
0A5F virgola
0A69 punto interrogativo
0A6D caratteri di controllo con operandi
0AD9 stampa carattere/i
0ADC memorizza indirizzo di linea, colonna e pixel
0B03 preleva parametri di posizione
0B24 stampa i diversi tipi di carattere
0B7F stampa tutti i caratteri
0BDB elabora il byte attributi
0C0A stampa un messaggio
0C3B gestione stampa ricorsiva
0C41 ricerca tabella
0C55 test per lo scroll
0CF8 messaggio 'scroll?'
0D4D attributi di colore temporanei
0D6B comando CLS
0DAF cancella l'intera display-area
0DD9 identificazione linea/colonna
0DFE routine di scroll
0E44 cancella la parte bassa dello schermo
0E88 gestione attributi
0E9B gestione schermo con numero di linea
0EAC COPY
0ECD invia il printer-buffer alla stampante
0EDF cancella il printer-buffer
0EF4 invio alla stampante di una linea
0F2C routine di editor per linea basic e per INPUT
0F81 aggiunge un car. alla linea in edit o input
0FA0 tabella tasti di editing
0FA9 controllo tasti di editing

0FF3 cursore in giu
1007 cursore a sinistra
100C cursore a destra
1015 DELETE
101E ignora i success.2 codici dall'input
1024 ENTER editing
1031 controllo cursore in editing
1059 cursore in su
1076 controllo codici SYMBOL SH. e GRAPHICS
107F routine di errore nell'editing
1097 cancella l'area di edit o la work-space a.
10A8 input da tastiera
111D stampa nella parte bassa dello schermo
1190 punta prima e ultima locaz. di edit-area o work-sp.
11A7 elimina da linea basic i formati fl.point nascosti

Routines esecutive

11B7 comando NEW
11CB entry-point principale (inizializzazione)
11DA controllo della RAM
12A2 ciclo principale di esecuzione
1391 tabella messaggi di errore
155D aggiunge una linea basic
15AF dati sui canali principali
15C6 dati di inizio sui flussi
15D4 attesa di un tasto
15E6 punta l'indirizzo di input
15EF routine principale di stampa
1601 apertura di canale
1615 selezione flag di canale
162D tabella codici di canale
1634 flag canale "K"
1642 flag canale "S"
164D flag canale "P"
1652 routine per fare spazio
1664 controllo dei puntatori
168F prelievo di un numero di linea
169E riserva spazio
16B0 reset al minimo ingombro di edit-area e seg.
16D4 richiesta di linea di edit
16DB "indexer" (scansione di tabelle)
16E5 comando CLOSE#

1701 CLOSE# di flussi associati ai canali K/S/P
1716 tabella per CLOSE#
171C rientro dalla subrout. CLOSE#
171E dati di flusso
1736 comando OPEN#
175D OPEN#/2, associa canale/flusso
177A tabella flusso OPEN#
1781 OPEN# "K"
1785 OPEN# "S"
1789 OPEN# "P"
1793 Usando CAT/ERASE/MOVE/FORMAT da' errore "O"
1795 auto-LIST
17F5 entry-point per LLIST
17F9 entry-point per LIST
1855 stampa un'intera linea di basic
18B6 test per l'identificatore di numero
18C1 stampa un carattere lampeggiante
18E1 stampa il cursore
190F prelievo numero di linea
1925 stampa caratteri in una linea basic
196E indirizzo di inizio di una linea
1980 confronto numeri di linea
1988 ricerca di un'istruzione
19B8 trova la prossima linea/variabile
19DD differenza
19E5 richiesta di spazio
19FB legge num.linea in edit-area
1A1B stampa numero di linea

Interpretazione di linee basic e di comandi

1A48 tab.sintassi: a)valori di offset per i com. basic
1A7A tab.sintassi: b)parametri (classi,separat., indirizzi)
1B17 analizzatore sintattico principale
1B28 loop istruzione
1B52 puntatore temporaneo per entry in tabella parametri
1B6F controllo separatore
1B76 return dopo corretta interpretazione
1B8A run a un dato numero di linea
1B9E trova indirizzo di linea a cui saltare
1BB2 gestione di REM
1BB3 controlla se la linea e' finita

1BBF controllo generale della linea
1BD1 vai alla linea successiva
1BEE controlla se l'istruzione e' terminata
1BF4 vai all'istruzione successiva
1C01 tabella della classe dei comandi
1C0D classi 00,03,05
1C16 salta alla giusta istruzione
1C1F classi 01,02,04
1C22 variabile in assegnazione
1C56 preleva un valore
1C6C classe 04
1C79 valutaz.espressione numerica/stringa
1C96 defin.colori permanenti (classe 07)
1CBE classe 09
1CDB classe 0B
1CDE preleva un numero
1CEE istruzione STOP
1CF0 istruzione IF
1D03 istruzione FOR
1D86 ricerca di DATA,DEF FN,NEXT
1DAB istruzione NEXT
1DDA controllo ciclo NEXT
1DEC istruzione READ
1E27 istruzione DATA
1E39 controllo presenza DATA/DEF.FN
1E42 istr. RESTORE
1E4F istr. RANDOMIZE
1E5F istr. CONTINUE
1E67 istr. GO TO
1E7A istr. OUT
1E80 istr. POKE
1E85 esamina due parametri sullo stack
1E94 trova gli interi
1EA1 istr. RUN
1EAC istr. CLEAR
1EED istr. GO SUB
1F05 controllo memoria disponibile
1F1A routine memoria libera
1F23 istr. RETURN
1F3A istr. PAUSE
1F54 tasto BREAK
1F60 istr. DEF FN
1FC3 ritorno anticipato da subroutine
1FC9 PRINT e LPRINT
1FF5 stampa un 'carriage return'
1FFC parametri di stampa

2045 fine stampa
204E controllo posizione di stampa
2070 modifica flusso
2089 istruzione INPUT
21B9 controllo dell'input
21D6 uso del canale "K"
21E1 parametri colore
226C gestione cambio colore
2294 istr. BORDER
22AA indirizzo pixel
22CB routine POINT
22DC istruzione PLOT
2307 due numeri da stack a BC
2314 un numero da stack ad A
2320 istr. CIRCLE
2382 istr. DRAW
247D definiz.param.iniziali (per CIRCLE e DRAW)
24B7 disegna una linea

Calcolo di espressioni

24FB routine di scansione
2530 controllo sintassi
2535 scansione SCREEN$
2580 scansione ATTR
2596 scansione tabella funzioni
25AF scansione routines funzioni
26C9 scansione routines variabili
2734 loop principale
2795 tabella operatori
27B0 tabella priorita'
27BD routine scansione FN
28AB skip-over (usato da FN)
28B2 ricerca in area variabili (per FN)
2951 ricerca argomento in stack funzioni
2996 controllo parametri in area variabili
2A52 routine slicing di stringa
2AB1 trasfer.a stack da A,B,C,D,E
2ACC calcolo prossima espress. in valore intero
2AEE esegue LD DE,(DE+1)
2AF4 esegue HL*DE (ritorna in HL)
2AFF istruz. LET
2BA6 numero da stack (o stringa da work-sp.) ad area variabili
2BAF seguito istruz. LET

2BC6 stringa in area variabili
2BF1 preleva LV (last value) da stack
2C02 istruz. DIM
2C88 test cifra o lettera
2C8D controllo validita' di un car.alfabetico
2C9B da formato decimale a floating point
2D1B controllo validita' di una cifra
2D22 controllo cifra/non cifra
2D28 da binario a floating point (reg.A)
2D2B da binario a floating point (reg.BC)
2D3B da intero a floating point

Routines aritmetiche

2D4F da formato xEy a floating point
2D7F preleva un intero da (HL)
2D8C memorizza un intero da (HL)
2DA2 da fl.point a formato compresso in BC
2DC1 calcola LOG(2^A)
2DD5 da fl.point ad A
2DE3 stampa un numero in formato fl.point
2F88 esegue CA=10*A+C
2F9B prepara un fl.point per addizione
2FBA preleva due numeri
2FDD shift a destra di num.fl.point
3004 addizione dei riporti
300F sottrazione
3014 addizione
30A9 esegue HL= HL*DE
30C0 prepara moltiplic. o divisione
30CA moltiplicazione
31AF divisione
3214 troncamento decimali
3293 riporta in stack due interi piccoli
3297 riporta in stack un numero fl.point

Il calcolatore floating point

32C5 tabella costanti
32D7 tabella indirizzi
335B esecuzione calcoli in fl.point
33A1 DELETE
33A2 singola operazione
33A9 controllo 5 byte di spazio

33B4 muove un fl.point allo stack
33C6 'literals' allo stack
33F7 salto delle costanti
3406 individua locaz.di memoria
340F prendi dall'area di memoria
341B una costante allo stack
342D invia all'area di memoria
343C scambia (i primi due dello stack)
3449 generatore di serie (di Chebyshev)
346A grandezza assoluta
346E manipolazione del segno
3492 funzione segno
34A5 funzione IN
34AC funzione PEEK
34B3 funzione USR
34BC funzione USR 'stringa' (per gli UDG)
34E9 test per zero
34F9 test per maggiore di zero
3501 operatore NOT
3506 test per minore di zero
350B test 'zero o uno'
351B operatore OR
3524 operat.AND tra due numeri
352D operat.AND tra numero e stringa
353B confronti <,<=,=,>,=>,<> numeri e stringhe
359C concatenazione di stringhe
35BF Hl e DE puntano lo stack
35C9 funzione CHR$
35DE funzioni VAl e VAL$
361F funzione STR$
3645 lettura dato
3669 funzione CODE
3674 funzione LEN
367A decrementa il contatore
3686 salto incondizionato
368F salto se (DE) e' vero
369B conclusione di un RST 0028
36A0 esegue il modulo di numero in stack
36AF funzione INT
36C4 calcolo di esponenziale
3713 funzione LN (logar.naturale)
3783 elaboraz.argomento di SIN o COS
37AA funzione COS
37B5 funzione SIN
37DA funzione TAN
37E2 funzione ARC TAN

3833 funzione ARC SIN
3843 funzione ARC COS
384A funzione SQR
3851 funzione EXP

386E-3CFF non usate (contengono FF)

3D00-3FFF set di caratteri (8 bytes per car.)

LIBRI DI SYSTEM -- libro mensile di Commodore Computer Club - Edizioni System Editoriale s.r.l. - v.le Famagosta, 75 - 20142 Milano - Tel. 02/8467348 - Dir. responsabile: Michele di Pisa - Reg. Trib. di Milano n. 370/82 - Sped. Abb. post. gr. III/70 - Stampa: Lito 3 - Cologno Monzese.

Lire 7.000
Anno 2 N. 5 - Giugno 1985
Distr. MePe

Tutti i segreti dello SPECTRUM